Chipmunk: Better Synchronized Multi-Signatures from Lattices

Nils Fleischhacker
Ruhr University Bochum
Bochum, Germany
mail@nilsfleischhacker.de

Gottfried Herold
Ethereum Foundation
Bonn, Germany
gottfried.herold@ethereum.org

Mark Simkin
Ethereum Foundation
Aarhus, Denmark
mark.simkin@ethereum.org

Zhenfei Zhang
Ethereum Foundation
Boston, USA
zhenfei.zhang@ethereum.org

ABSTRACT

Multi-signatures allow for compressing many signatures for the same message that were generated under independent keys into one small aggregated signature. This primitive is particularly useful for proof-of-stake blockchains, like Ethereum, where the same block is signed by many signers, who vouch for the block's validity. Being able to compress all signatures for the same block into a short string significantly reduces the on-chain storage costs, which is an important efficiency metric for blockchains.

In this work, we consider multi-signatures in the synchronized setting, where the signing algorithm takes an additional time parameter as input and it is only required that signatures for the same time step are aggregatable. The synchronized setting is simpler than the general multi-signature setting, but is sufficient for most blockchain related applications, as signers are naturally synchronized by the length of the chain.

We present Chipmunk, a concretely efficient lattice-based multi-signature scheme in the synchronized setting that allows for signing an a-priori bounded number of messages. Chipmunk allows for non-interactive aggregation of signatures and is secure against rogue-key attacks. The construction is plausibly secure against quantum adversaries as our security relies on the assumed hardness of the short integer solution problem.

We significantly improve upon the previously best known construction in this setting by Fleischhacker, Simkin, Zhang (CCS 2022). Our aggregate signature size is 5× smaller and for 112 bits of security our construction allows for compressing 8192 individual signatures into a multi-signature of size less than 200 KB. We provide a full implementation of Chipmunk and provide extensive benchmarks studying our construction's efficiency.

CCS CONCEPTS

• **Security and privacy → Digital signatures**.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
CCS '23, November 26–30, 2023, Copenhagen, Denmark
© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0050-7/23/11…$15.00
https://doi.org/10.1145/3576915.3623219

KEYWORDS

multi-signatures, lattice based cryptography

ACM Reference Format:
Nils Fleischhacker, Gottfried Herold, Mark Simkin, and Zhenfei Zhang. 2023. Chipmunk: Better Synchronized Multi-Signatures from Lattices. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS '23), November 26–30, 2023, Copenhagen, Denmark.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3576915.3623219

1 INTRODUCTION

Multi-signatures [18, 24] allow for compressing distinct signatures for the same message generated by different signers into one small aggregated signature. Such signature schemes are a powerful tool in distributed systems, like blockchains, where parties vouch for the validity of messages on the network by signing them. Rather than storing an amount of signatures that is linear in the number of parties that vouched for a specific messages, multi-signatures allow for storing a much shorter string that vouches for a message on behalf of all signers simultaneously. Popular proof-of-stake blockchains like Ethereum[1] and DFinity[2] employ multi-signatures at the core of their consensus layer.

The most popular multi-signature scheme used in practice is a construction due to Boneh, Gentry, Lynn, and Shacham [5] based on a signature scheme due to Boneh, Lynn, and Shacham (BLS) [7]. Their resulting multi-signatures are extremely small, but the security of their construction relies on the assumed hardness of computing discrete logarithms over pairing-friendly groups. It was shown by Shor [27] that the discrete logarithm problem can be solved efficiently by quantum computers, meaning that any cryptographic primitive basing its security on such an assumption is insecure in the presence of a quantum adversary.

Luckily, not all computational hardness assumptions are created equal and some seem to remain hard in the presence of quantum adversaries. Building multi-signatures from computational hardness assumptions that withstand quantum adversaries is both a theoretically and practically important question. While it may not be clear when practically relevant quantum computers will appear, it is important to have secure alternatives for important cryptographic primitives ahead of time.

One class of cryptographic hardness assumptions that seems to be particularly resilient against quantum adversaries is lattice-based

[1] https://github.com/ethereum/annotated-spec/blob/master/phase0/beacon-chain.md#attestation
[2] https://dfinity.org/whitepaper.pdf

cryptography. Two of the three post-quantum signature schemes that were selected for standardization by NIST in 2022 base their security on hardness assumptions related to lattices and, not surprisingly, there has also been significant interest in constructing multi-signatures from lattice hardness assumptions [8, 9, 11, 13–15, 19, 22, 26]. The current multi-signature constructions, however, do still have significant drawbacks that hinder their practical deployment. The constructions of El Bansarkhani and Sturm [11] and Ma and Jiang [22] assume that the keys of all signers are generated honestly. This is not a realistic assumption as an adversarial signer could aim to perform a rogue-key attack by generating a malformed verification key that depends on honest signers' keys and allows for forging aggregated signatures, which falsely claim that both the malicious and the honest parties signed a message that was not actually signed by them. The scheme of Kansal and Dutta [19] was shown to be insecure by Liu et al. [20]. The constructions of Fukumitsu and Hasegawa [14, 15], Ma and Jiang [22], and Peng and Du [26], and Boschini, Takahashi, and Tibouchi [8] all require interaction between the signers for generating a joint multi-signature. Such an interaction between independent signers is difficult to realize in many distributed systems as the signers may be online at different times and may even not know of each others existence.

Recently, Fleischhacker, Simkin, and Zhang [13] presented a lattice-based multi-signature construction named Squirrel, which allows for non-interactive aggregation and is secure against rogue-key attacks. They consider a simplified setting, where signer's keys are only able to sign an a-priori bounded number of messages and where signers are synchronized in the sense that aggregation only has to work for signatures that were generated for the same time step and same message. This simplified setting is still sufficiently strong for most blockchain applications, where signers do not sign more than one message per block and are naturally synchronized by the length of the current chain. While an a-priori bound on the number of messages that can be signed may seem like a strong limitation, one can simply set this number large enough, e.g. to 2^{24} which would allow a signer to sign a message every 10 seconds for 5 years non-stop. Aiming for 112 bits of security, their individual signatures are roughly 50 KB large and aggregating 4096 signatures results in a multi-signature that are 771 KB large.

Squirrel represents a significant step forward for multi-signature schemes that are plausibly secure in the presence of a quantum adversary and are concretely efficient. For real-world practical scenarios their aggregated signatures seem, however, still too large to be really used. As a point of reference, a full Ethereum block is on average less than 130 KB large[3], which would mean that one block could not even fit a single multi-signature.

1.1 Our Contribution

In this work we present Chipmunk[4], a multi-signature scheme in the synchronized setting [1, 10, 16, 17] with an a-priori bound on the number of signatures that can be issued per key. We aim for the exact same setting as Squirrel [13], but provide both theoretical and practical improvements.

On the theoretical side, we strengthen the security notions for multi-signatures by requiring that aggregation involving malformed but verifying adversarial signatures will succeed with high probability. In Squirrel, aggregation was only required to work for honestly generated individual signatures. In principle, their security model would allow an adversary to perform a denial-of-service attack against the signature aggregation procedure by providing a single verifying, but malformed signature. In a real-world distributed system, such an attack on liveness would be highly problematic. We strengthen their security definitions to formally ensure that successfully verifying individual signatures will be successfully aggregated, even if they are chosen maliciously.

On the practical side, our scheme Chipmunk produces smaller individual and aggregated signatures, when compared to Squirrel. In terms of computational efficiency metrics, Chipmunk either significantly outperforms Squirrel or remains comparable in speed. In terms of bandwidth, Chipmunk's aggregate signatures are smaller by a factor of 4.8×, when compared to Squirrel. For keys that can generate 2^{24} signatures, an individual Chipmunk signatures is 50 KB and aggregating 8192 signatures results in an aggregate signature that is 180 KB large.

We have fully implemented Chipmunk and provide extensive benchmarks and comparisons to Squirrel in Section 6. Chipmunk currently is the most concretely efficient multi-signature known that is based on assumptions that are assumed to remain valid in the presence of a quantum adversary.

1.2 Technical Overview

Conceptually, Chipmunk closely follows the blueprint that was introduced by Fleischhacker, Simkin, and Zhang [13]. Recall that in their work and in ours we only aim to issue an a-priori bounded number of signatures, meaning that key generation is parameterized by τ and produces a public key that can be used to sign 2^τ messages. Further recall that we are in the synchronized setting, meaning that we only aim to aggregate signatures for the same message that were issued at the same time step.

In Squirrel, each signer's public key pk is a homomorphic vector commitment of length 2^τ, where position i commits to pk^i, which is the public key of a key-homomorphic one-time signature scheme. To sign message m at time step i, the signer opens the commitment pk to pk^i at position i and uses the corresponding one-time signing key to sign the message m. The signature is a vector itself that consists of pk^i, the corresponding opening, and the signature of m under this one-time key. To verify that a message m was signed for time step i, the verifier checks that the given public key pk^i is a valid opening of the i-th position of the corresponding signer's public key pk and that the given signature verifies for message m under the public key pk^i.

Aggregation of such signatures is performed by exploiting the homomorphic properties of the vector commitment and the one-time signature scheme. To aggregate signatures, roughly speaking, one simply adds up all the individual commitment openings, the one-time keys, and the corresponding one-time signatures. The homomorphism of the vector commitment scheme ensures that the sum of openings is a valid opening for the sum of committed

[3] https://etherscan.io/chart/blocksize
[4] Smaller than squirrels, cuter than squirrels.

messages, i.e. the one-time public keys, under the sum of commitments. The key-homomorphic property of the one-time signature scheme ensures that the sum of signatures for the same message verifies under the sum of one-time public keys. Chipmunk follows this blueprint, but improves upon all building blocks that are being used and thereby significantly reduces the multi-signature size of Chipmunk.

Key-Homomorphic One-Time Signatures. Squirrel relies on a key-homomorphic one-time signature scheme that is similar to those of Boneh and Kim [6] and Lyubashevsky and Micciancio [21]. The details of the construction are not relevant for now. For Chipmunk, we use almost the exact same scheme, but observe by carefully inspecting their original security proof that a minor modification of the construction used in Squirrel allows for the proof to produce much tighter parameters and thus smaller signatures.

Homomorphic Vector Commitments. The vector commitment used by Squirrel is a homomorphic analogue of the classical Merkle tree construction. To make a Merkle tree homomorphic, the idea is to employ a homomorphic hash function to compute the node's values of the tree. Now when adding two trees node-wise, one obtains a new valid tree. Ajtai [2] introduced such a homomorphic hash function based on the short integer solution problem. The main difficulty with using this hash function is the fact that hash output values need to be transformed into valid hash input values in a way that is efficient and maintains the homomorphic properties we would like our tree to have. Without going into the details, the problem is that inputs for Ajtai's hash function need to have small norm, but outputs have potentially very large norms. Fleischhacker, Simkin, and Zhang [13] solved this problem by effectively performing a binary decomposition of the hash function's output values, which resulted in vectors with infinity norm one, which could then again be used as hash function inputs. In Chipmunk, we generalize their trick of decomposing values into binary vectors to decomposition into vectors of small norm. While conceptually simple, we show that this change allows us to significantly reduce the size of our homomorphic vector commitment openings.

Chipmunk Multi-Signatures. Our construction of multi-signatures from vector commitments and one-time signatures follows the blueprint that was already outlined above. One thing we glossed over so far are rogue-key attacks. If we were to simply add up individual signatures, then our scheme would be susceptible to an adversary that first sees the honest parties public keys and then generates a malicious public key that allows for forging multi-signatures involving honest signers for arbitrary messages. To avoid this type of attack, the individual signatures are multiplied by randomizer values before being added up. These randomizing values need to be from a space that is large enough to be unpredictable for the adversary, but cannot be too large as the multi-signature scheme's efficiency would deteriorate. In Squirrel, aggregation was a one-attempt process. In Chipmunk, we repeatedly choose fresh randomization values and attempt aggregation until the aggregated signature is "small enough". We show that doing this alllows us to reduce signature size without affecting the security or the performance of our construction.

We note that we grossly oversimplified many things in our above overview and that the precise construction and our improvements are more technically involved. While each individual improvement may seem conceptually simple, they all add up to significantly improved signature sizes, resulting in the by far most efficient multi-signature scheme from lattice assumptions to date.

2 PRELIMINARIES

This section introduces notation, some basic definitions and a few basic lemmas that we will use throughout this work. We denote by $\lambda \in \mathbb{N}$ the security parameter and by $\mathrm{poly}(\lambda)$ any function that is bounded by a polynomial in λ. A function f in λ is negligible, if for every $c \in \mathbb{N}$, there exists some $N \in \mathbb{N}$, such that for all $\lambda > N$ it holds that $f(\lambda) < 1/\lambda^c$. We denote by $\mathrm{negl}(\lambda)$ any negligible function. An algorithm is PPT if it is modeled by a probabilistic Turing machine with a running time bounded by $\mathrm{poly}(\lambda)$.

Let S be a set. We write $x \leftarrow S$ for the process of sampling an element of S uniformly at random. Let T be a full binary tree of depth d. We denote the root node of T by the empty string ϵ, and for any node v, $v\|0$ and $v\|1$ denotes the left and right child of v respectively. In particular, $\{0,1\}^d$ is the set of leaves of T. A labeled full binary tree with labels in S is represented by a labeling function $\mathrm{label}\colon \{0,1\}^{\leq d} \to S$.

Let $\boldsymbol{v}, \boldsymbol{u}$ be vectors of length m. We throughoutly use 1-based indices in this work. We write $\boldsymbol{v}^\intercal$ to denote the transpose of $\boldsymbol{v}$ and v_i to denote the i-th entry in the vector for $1 \leq i \leq m$. We generalize this notation and write $\boldsymbol{v}_{<i}$ to denote the $(i-1)$-length prefix of $\boldsymbol{v}$. We use the same notation for a bit-string s, denoting by s_i the i-th bit and by $s_{<i}$ the prefix consisting of the first $i-1$ bits of s. For $0 \leq t \leq 2^\tau - 1$ we denote by $\mathrm{bin}_\tau(t) \in \{0,1\}^\tau$ the big-endian binary decomposition of t (possibly with leading zeros to ensure a fixed length of τ). For $n \in \mathbb{N}$, we denote by $[n]$ the set $\{1, \ldots, n\}$.

Our concrete construction works over a power-of-two cyclotomic polynomial ring. Let $\Phi_{2n} = X^n + 1$ be the cyclotomic polynomial with n a power of 2. We work in the polynomial ring $\mathcal{R} = \mathbb{Z}[X]/\langle X^n + 1\rangle$. For the purpose of taking norms and transmitting data, we represent elements of $\mathcal{R}$ as n-dimensional vectors $\mathbb{Z}^n$ with $(c_1, \ldots, c_n)^\intercal \in \mathbb{Z}^n$ representing the ring element $\sum_{i=1}^n c_i X^{i-1}$. For any odd prime number q, we always represent $\mathbb{Z}_q$ by the set $\{-\frac{q-1}{2}, \ldots, \frac{q-1}{2}\}$ centered around 0. For an odd prime q, we denote by $\mathcal{R}_q$ the quotient ring of $\mathcal{R}$ modulo q, represented by vectors in $\mathbb{Z}_q^n$. Whenever we need to take representatives to view these as elements from $\mathcal{R}$, we do so by taking representatives centered around 0 as above. For efficiency reasons, our parameter choice will always satisfy $q \equiv 1 \bmod 2n$, so we can use more efficient NTT-based multiplication in $\mathcal{R}_q$. Let $\boldsymbol{c} \in \mathcal{R}$ be a ring element with coefficients $(c_1, \ldots, c_n)$. We work, unless specified otherwise, with the $\|.\|_\infty$-norm. We define $\|\boldsymbol{c}\| := \|\boldsymbol{c}\|_\infty = \max_i |c_i|$ and $\|\boldsymbol{c}\|_1 = \sum_i |c_i|$ on $\mathcal{R}$ by taking the norm of the coefficient vector in the monomial basis. We extend these definitions to norms on $\mathcal{R}_q$ by taking representatives in $\mathcal{R}$ using coefficients in $\{-\frac{q-1}{2}, \ldots, \frac{q-1}{2}\}$. We also extend the definition of $\|.\|_\infty$ to $\mathcal{R}^m$ for any m by $\|\boldsymbol{c}\|_\infty = \max_i \|\boldsymbol{c}_i\|_\infty$.

Our convention is that mixed multiplication of an element from $\mathcal{R}$ with an element from $\mathcal{R}_q$ gives an element from $\mathcal{R}_q$, thereby viewing $\mathcal{R}_q$ as an $\mathcal{R}$-module (see Definition 2.3 below).

We denote by $\mathcal{B}_{\beta,q}$ the ball $\mathcal{B}_{\beta,q} = \{a \in \mathcal{R}_q \mid \|a\| \leq \beta\}$. We are only interested in the case $\beta < \frac{q}{2}$. By $\mathcal{T}_\alpha = \{a = (a_1 + a_2 \cdot X + \cdots + a_n X^{n-1}) \in \mathcal{R} \mid \|a\|_\infty \leq 1 \wedge \sum_{i=1}^n |a_i| = \alpha\}$ we denote the

set of polynomials with ternary coefficients, i.e. coefficients from $\{-1, 0, 1\}$, and with exactly α non-zero coefficients.

Observe that for our choices of ring $\mathcal{R}$ and norm, for any $a \in \mathcal{R}$, we have $\|a\| = \|X \cdot a\|$, because multiplication by X acts on the coefficient vector as a cyclic shift (up to sign). For such rings and norms, we can make use of the following simple lemma that allows us to bound the norm of the product of two polynomials.

Lemma 2.1 ([25]). *Let $a, b \in \mathcal{R}$ be two polynomials. Then $\|b \cdot a\| \leq \|a\|_1 \cdot \|b\|$.*

The security of our constructions relies on the hardness of the short integer solution problem defined over rings as follows.

Definition 2.2 (Ring Short Integer Solution Problem). For a ring $\mathcal{R}$ and parameters $\mu, q, \beta \in \mathbb{N}$, the $\mathsf{SIS}_{\mathcal{R},q,\mu,\beta}$ problem is hard if for all PPT algorithms $\mathcal{A}$ it holds that

$$\Pr[\boldsymbol{a} \leftarrow \mathcal{R}_q^{\mu}; \boldsymbol{s} \leftarrow \mathcal{A}(\boldsymbol{a}) : \boldsymbol{s} \in \mathcal{B}_{\beta,q}^{\mu} \setminus \{\boldsymbol{0}\} \wedge \boldsymbol{a}^{\intercal}\boldsymbol{s} = 0] \leq \mathsf{negl}(\lambda)$$

$\mathcal{R}$-modules. In order to aggregate signatures, we will be taking linear combinations of individual elements, where for security reasons the coefficients need to be from a sufficiently large space. We use the ring $\mathcal{R} = \mathbb{Z}[X]/\langle X^n + 1\rangle$ for those coefficients. This means that for both signatures and certain intermediate objects appearing during our constructions, we need to be able to both add them together and to multiply them with elements from $\mathcal{R}$. Recall that this is precisely captured by the notion of an $\mathcal{R}$-module, so let us recall some relevant notions here for convenience to the reader.

Definition 2.3 ($\mathcal{R}$-module). For a commutative ring $\mathcal{R}$, an $\mathcal{R}$-module A is an abelian group (with addition denoted by $+$) together with a multiplication operation

$$\cdot_A : \mathcal{R} \times A \to A, \quad (r, x) \mapsto r \cdot_A x$$

satifying $(rs) \cdot_A x = r \cdot_A (s \cdot_A x)$ (associativity), $(r + s) \cdot_A x = (r \cdot_A x) + (s \cdot_A x)$ as well as $r \cdot_A (x + y) = (r \cdot_A x) + (r \cdot_A y)$ (distributivity) and $1 \cdot_A x = x$ for all $r, s \in \mathcal{R}, x, y \in A$.

This is really the same definition as a vector space over a field, except that we use a ring instead of a field. As opposed to (finite-dimensional) vector spaces, not every (finitely generated) $\mathcal{R}$-module is isomorphic to $\mathcal{R}^n$ for some n. Similar to vector spaces, the multiplication is often denoted by just $\cdot$ or even just concatenation; we only write $\cdot_A$ here for emphasis.

We will only consider $\mathcal{R}$-modules for the specific choice of ring $\mathcal{R} = \mathbb{Z}[X]/\langle X^n + 1\rangle$. We will only need $\mathcal{R}$-modules of the form $\mathcal{R}^n$ or $\mathcal{R}_q^n$ for q prime and $n \in \mathbb{N}$ with module structures given by

$$\cdot_{\mathcal{R}^n} : \mathcal{R} \times \mathcal{R}^n \to \mathcal{R}^n, \quad (r, (x_1, \ldots, x_n)) \mapsto (r_1 x_1, \ldots, r_n x_n)$$

and

$$\cdot_{\mathcal{R}_q^n} : \mathcal{R} \times \mathcal{R}_q^n \to \mathcal{R}_q^n,$$
$$(r, (x_1, \ldots, x_n)) \mapsto (r_1 x_1 \bmod q, \ldots, r_n x_n \bmod q)$$

respectively. It is straightforward to check that these are $\mathcal{R}$-modules.

We also need appropriate maps between modules that preserve this structure. Notably, a map $f : A \to B$ between $\mathcal{R}$-modules with the same $\mathcal{R}$ is called $\mathcal{R}$-linear, if $f(x + y) = f(x) + f(y)$ and $f(rx) = rf(x)$ holds for each $x, y \in A, r \in \mathcal{R}$. The composition of $\mathcal{R}$-linear maps gives a $\mathcal{R}$-linear map. Examples of $\mathcal{R}$-linear maps are the modular reduction map $\bmod q : \mathcal{R} \to \mathcal{R}_q$ and maps of the form $\mathcal{R}^n \to \mathcal{R}^m, \boldsymbol{v} \mapsto A \cdot \boldsymbol{v}$ for a fixed matrix $A \in \mathcal{R}^{m \times n}$. The former follows from compatibility of modular reduction with $+$ and $\cdot$. The latter holds because $A(\boldsymbol{v} + \boldsymbol{w}) = A\boldsymbol{v} + A\boldsymbol{w}$ and $Ar\boldsymbol{v} = rA\boldsymbol{v}$ for all $\boldsymbol{v}, \boldsymbol{w} \in \mathcal{R}^n, r \in \mathcal{R}, A \in \mathcal{R}^{m \times n}$, since $\mathcal{R}$ is commutative.

Norm growth. To ensure that during aggregation, our norms don't grow too much, we will use the following auxilliary lemma to control the growth of the norm bounds.

Lemma 2.4. *Let n, α_w, ρ, β be positive integers such that n is a power of two. Let $\mathcal{R}$ be the polynomial ring $\mathbb{Z}_q[X]/\langle X^n + 1\rangle$. Then for any $\ell \leq \rho$, for any $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_\ell \in \mathcal{R}$ with $\|\boldsymbol{x}_i\| \leq \beta$ and any growth factors $\zeta \geq 1$, we have*

$$\Pr\left[w_1, \ldots, w_\ell \leftarrow \mathcal{T}_{\alpha_w} : \left\|\sum_{i=1}^{\ell} w_i \cdot \boldsymbol{x}_i\right\| > \zeta \cdot \beta\right] < 2n \cdot \exp\left(-\frac{\zeta^2}{2\alpha\rho}\right) .$$

Proof. Recall that $\mathcal{T}_{\alpha_w}$ denotes ternary polynomials with weight exactly α_w. We will show that the claim holds even if we fix the positions of the non-zero entries in each $\boldsymbol{w}_i$ and only consider the randomness coming from the ± 1-signs. Now observe that for each fixed k, the kth coefficient y_k of $\sum_{i=1}^{\ell} w_i \boldsymbol{x}_i$ is a sum of the form

$$y_k = \sum_{j=1}^{\alpha_w \cdot \ell} b_j c_j \ ,$$

where each b_j is some coefficient of some $\boldsymbol{x}_i$ and each $c_j \in \{-1, +1\}$ iid, corresponding to a sign choice of some coefficient of some $\boldsymbol{w}_i$ (everything depending on k). Thus, the expected value of y_k is 0 and $|b_j| \leq \beta$, so changing any individual c_j out of the $\ell\alpha_w$ many c_j's can change the value of y_k by at most 2β. We can thus apply McDiarmid's inequality [23] to obtain

$$\Pr\left[|y_k| > \zeta\beta\right] \leq 2\exp\left(\frac{-2(\zeta\beta)^2}{\ell\alpha_w(2\beta)^2}\right) \leq 2\exp\left(-\frac{\zeta^2}{2\alpha_w\rho}\right) .$$

Taking a union bound over all n coefficients y_k of $\sum_{i=1}^{\ell} \boldsymbol{w}_i \boldsymbol{x}_i$ then gives the claim. □

We will construct a particular homomorphic vector commitment (HVC) denoted by $\mathsf{HVC}^{\mathsf{Chip}}$ in Section 3. In Section 4, we construct a key-homomorphic one-time signature scheme (KOTS) denoted by $\mathsf{KOTS}^{\mathsf{Chip}}$. In Section 5, we combine those components to construct a synchronized aggregatable signature scheme. Our concrete constructions depends on a significant number of tunable parameters, whose choices affects both security, efficiency and functionality. Table 1 gives an overview and is intended as a reference for later, to aid the reader.

3 HOMOMORPHIC VECTOR COMMITMENTS

In this section, we define and instantiate homomorphic vector commitments, which allow for committing to a long vector with a short commitment value. Positions in the vector can be individually opened using a short opening value. We follow the definitions for vector commitments of Fleischhacker, Simkin, and Zhang [13], but we require somewhat different and incomparable homomorphic properties. The definition of [13] only requires honestly generated commitments to have homomorphic properties, whereas our definition requires the homomorphism to work for any individually

Parameter	Meaning
n	Dimension of the ring $\mathcal{R} = \mathbb{Z}[X]/\langle X^n + 1\rangle$ we are working over; n is a power of 2.
q	Prime number for HVC. Our HVC internally works modulo q, i.e. with $\mathcal{R}_q = \mathcal{R}/\langle q\rangle$.
q'	Prime number for KOTS. Our KOTS works modulo q', i.e. with $\mathcal{R}_{q'} = \mathcal{R}/\langle q'\rangle$. The HVC is used to commit to elements of $\mathcal{R}_{q'}$.
τ	Depth of our Merkle tree. 2^τ is the number of indices for the HVC. This is also the number of time slots for the synchronized multi-signature.
ρ	Maximum number of homomorphic vector commitments or signatures that we support aggregating.
η	Arity parameter. Our constructions make use of $2\eta + 1$-ary decomposition.
κ	Number of limbs used in the decomposition of $\mathcal{R}q$ elements.
κ'	Number of limbs used in the decomposition of $\mathcal{R}q'$ elements.
ξ	Dimension (over $\mathcal{R}_{q'}$) of the elements our HVC commits to.
γ	Dimension (over $\mathcal{R}_{q'}$) of public parameters and secret key components in the KOTS.
β_{agg}	Norm bound for HVC after aggregation/homomorphic addition.
β_σ	Norm bound for KOTS signatures after aggregation/homomorphic addition.
α_w	Hamming weight for ring elements used as coefficients for homomorphic addition of our HVC or KOTS.
α_H	Hamming weight for ring elements used as randomizers in the construction of individual KOTS signatures.
φ	Norm bound parameter for secret keys of the KOTS.
ε	Error bound. Individual aggregation attempts may fail with at most this probability for our HVC or KOTS.
χ	Maximum number of aggregation attempts. Aggregation ultimately fails if χ individual attempts have failed.

Table 1: Parameters used in our concrete homomorphic vector commitment (HVC) and key-homomorphic one-time signature (KOTS) scheme. Since we combine those to a synchronized aggregatable signature scheme later, the parameters are related.

verifying commitments and openings. On the other hand, [13] requires that the homomorphism works with probability 1, whereas we allow some noticeable error. Among other things, this modification of the definition allows us to instantiate homomorphic vector commitments more compactly.

Definition 3.1. Let $\tau \in \mathbb{N}$ be fixed. Let $\mathcal{R}$ be a ring and let A_{dom}, A_{com}, A_{op} be $\mathcal{R}$-modules. A homomorphic vector commitment scheme (HVC) for domain A_{dom} and vectors of length 2^τ is defined by six PPT algorithms $\text{HVC} = (\text{Setup}, \text{Com}, \text{Open}, \text{iVf}, \text{sVf}, \text{wVf})$.

- $\text{pp} \leftarrow \text{Setup}(1^\lambda)$ The setup algorithm takes as input the security parameter and outputs public parameters.
- $c \leftarrow \text{Com}(\text{pp}, \boldsymbol{m})$ The commitment algorithm gets as input the public parameters and a vector $\boldsymbol{m} \in A_{\text{dom}}^{2^\tau}$ and outputs a commitment $c \in A_{\text{com}}$.
- $d \leftarrow \text{Open}(\text{pp}, c, \boldsymbol{m}, t)$ The opening algorithm gets as input the public parameters, a commitment, the committed vector, and an index and outputs a decommitment $d \in A_{\text{op}}$.
- $\boldsymbol{m}/\perp \leftarrow \text{iVf}(\text{pp}, c, t, d)$ The individual verification algorithm takes as input public parameters, a commitment, an index, and a decommitment and outputs either $\boldsymbol{m} \in A_{\text{dom}}$ or an error symbol.
- $\boldsymbol{m}/\perp \leftarrow \text{sVf}(\text{pp}, c, t, d)$ The strong verification algorithm has the same input and output domains as the individual verification algorithm.
- $\boldsymbol{m}/\perp \leftarrow \text{wVf}(\text{pp}, c, t, d)$ The weak verification algorithm has the same input and output domains as the individual verification algorithm.

For our purposes, $\mathcal{R}$ will always be $\mathcal{R} = \mathbb{Z}[X]/\langle X^n + 1\rangle$ for n a power of 2, as in Section 2. Our domain, commitment and opening space will always be of the form $A_{\text{dom}} = \mathcal{R}_{q'}^{\ell_{\text{dom}}}$, $A_{\text{com}} = \mathcal{R}_q^{\ell_{\text{com}}}$, $A_{\text{op}} = \mathcal{R}^{\ell_{\text{op}}}$ for some primes q, q'. Note that (correctly verifying) decommitments $d \in A_{\text{op}}$ will have small coefficients and undergo arithmetic modulo q, so the reader may think of them as elements from $\mathcal{R}_q^{\ell_{\text{op}}}$, as Squirrel [13] does.

We can easily generalize this definition slightly and have A_{com} and A_{op} depend on the particular choice of $\text{pp} \leftarrow \text{Setup}(1^\lambda)$, but will do not need that.

Definition 3.2 (Individual Correctness). Let HVC be a vector commitment scheme for domain A_{dom} and vector length 2^τ. HVC is individually correct, if for all security parameters $\lambda \in \mathbb{N}$, vectors $\boldsymbol{m} \in A_{\text{dom}}^{2^\tau}$, indices $1 \le t \le 2^\tau$, parameters $\text{pp} \leftarrow \text{Setup}(1^\lambda)$, commitments $\boldsymbol{c} \leftarrow \text{Com}(\text{pp}, \boldsymbol{m})$, and openings $\boldsymbol{d} \leftarrow \text{Open}(\text{pp}, \boldsymbol{c}, \boldsymbol{m}, t)$ it holds that

$$\text{iVf}\Big(\text{pp}, \boldsymbol{c}, t, \boldsymbol{d}\Big) = \boldsymbol{m}_t \quad .$$

We require that individually verifying commitments and their respective decommitments can be homomorphically aggregated by computing a random $\mathcal{R}$-linear combination of them. Such aggregated commitments and decommitments should still *strongly* verify with high probability over the choice of the random linear combination, provided the coefficients of the linear combination are from some restricted subset (such as small elements).

Definition 3.3 (Probabilistic Homomorphism). Let HVC be a vector commitment scheme for domain A_{dom} and vector length 2^τ. Let $\rho \in \mathbb{N}$, $0 \le \varepsilon \le 1$ and $W \subseteq \mathcal{R}$. HVC is (ρ, W, ε)-probabilistically homomorphic, if for all security parameters $\lambda \in \mathbb{N}$, number of aggregated commitments $\ell \le \rho$, indices $1 \le t \le 2^\tau$, parameters $\text{pp} \leftarrow \text{Setup}(1^\lambda)$, commitments $\boldsymbol{c}^i \in A_{\text{com}}$, and decommitments

$\boldsymbol{d}^i \in A_{\mathsf{op}}$ with $\mathsf{iVf}(\mathsf{pp}, \boldsymbol{c}^i, t, \boldsymbol{d}^i) = \boldsymbol{m}^i$ such that $\boldsymbol{m}^i \neq \perp$ it holds that

$$\Pr\left[w^1, \ldots, w^\ell \leftarrow W : \mathsf{sVf}\Big(\mathsf{pp}, \sum_{i=1}^{\ell} w^i \boldsymbol{c}^i, t, \sum_{i=1}^{\ell} w^i \boldsymbol{d}^i\Big) = \sum_{i=1}^{\ell} w^i \boldsymbol{m}_t^i\right] \geq 1 - \varepsilon \ .$$

We additionally require that a further limited homomorphism still holds, even for maliciously *aggregated* commitments. For any two, even maliciously generated, commitments and their two respective openings that *strongly* verify, their difference will still *weakly* verify.

Definition 3.4 (Robust Homomorphism). Let HVC be a vector commitment scheme for domain A_{dom} and vector length 2^τ. HVC is robustly homomorphic if for all security parameters $\lambda \in \mathbb{N}$, public parameters $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, indices $1 \leq t \leq 2^\tau$, (possibly malformed) commitments $\boldsymbol{c}^0, \boldsymbol{c}^1 \in A_{\mathsf{com}}$, and (possibly malformed) decommitments $\boldsymbol{d}^0, \boldsymbol{d}^1 \in A_{\mathsf{op}}$ with

$$\mathsf{sVf}(\mathsf{pp}, \boldsymbol{c}^0, t, \boldsymbol{d}^0) = \boldsymbol{m}^0 \quad \text{and} \quad \mathsf{sVf}(\mathsf{pp}, \boldsymbol{c}^1, t, \boldsymbol{d}^1) = \boldsymbol{m}^1$$

such that $\boldsymbol{m}^0, \boldsymbol{m}^1 \neq \perp$ it holds that

$$\mathsf{wVf}(\mathsf{pp}, \boldsymbol{c}^0 - \boldsymbol{c}^1, t, \boldsymbol{d}^0 - \boldsymbol{d}^1) = \boldsymbol{m}^0 - \boldsymbol{m}^1 \ .$$

Finally, we require the commitments to be position binding.

Definition 3.5 (Position-Binding). Let HVC be a vector commitment scheme. HVC is position binding if for all security parameters λ and all PPT algorithms $\mathcal{A}$ it holds that

$$\Pr\left[\begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda); \\ (\boldsymbol{c}, t, \boldsymbol{d}_0, \boldsymbol{d}_1) \leftarrow \mathcal{A}(\mathsf{pp}); \\ \boldsymbol{m}_0 \leftarrow \mathsf{wVf}(\mathsf{pp}, c, t, d_0); \\ \boldsymbol{m}_1 \leftarrow \mathsf{wVf}(\mathsf{pp}, c, t, d_1) \end{array} : \boldsymbol{m}_0 \neq \boldsymbol{m}_1 \wedge \perp \notin \{\boldsymbol{m}_0, \boldsymbol{m}_1\}\right] \leq \mathsf{negl}(\lambda) \ .$$

3.1 Squirrel's Homomorphic Vector Commitment

Since our homomorphic vector commitment is strongly based on Squirrel [13], we recap their construction, albeit informally, in a bit more detail. Somewhat simplified, this commits to 2^τ (small) entries from $A_{\mathsf{dom}} = \mathcal{R}_q^{\ell_{\mathsf{dom}}}$ by using a Merkle tree with a homomorphic hash function. If we naively build a Merkle tree, this would mean that we construct a complete binary tree with 2^τ leaves, where each leaf corresponds to an entry we want to commit to. To each non-leaf node v, we associate the hash of its child nodes. Concretely, the hash function utilized is Ajtai's hash function [2], which hashes child nodes $\boldsymbol{c}_1, \boldsymbol{c}_2 \in \mathcal{R}_q^{\ell_{\mathsf{dom}}}$ to

$$h_{\mathsf{Ajtai}}(\boldsymbol{c}_1, \boldsymbol{c}_2) \coloneqq \boldsymbol{a}_1^\intercal \boldsymbol{c}_1 + \boldsymbol{a}_2^\intercal \boldsymbol{c}_2 \bmod q$$

using a uniformly random $\mathcal{R}_q$-linear map given by fixed public uninform $\boldsymbol{a}_1, \boldsymbol{a}_2 \leftarrow \mathcal{R}_q^{\ell_{\mathsf{dom}}}$. Now, setting the relationship between parent node $\boldsymbol{p}$ and child nodes $\boldsymbol{c}_1, \boldsymbol{c}_2$ as $\boldsymbol{p} = h_{\mathsf{Ajtai}}(\boldsymbol{c}_1, \boldsymbol{c}_2)$ does not quite work: firstly, the range of the hash function is not A_{dom}, which prevents iterating this construction. Secondly, this hash function is only binding (based on some appropriate ring-SIS assumption) if we restrict its input to small elements. To solve these issues, Squirrel chooses a second (public, fixed) linear function

$$\mathsf{proj} \colon \mathcal{R}_q^{\ell_{\mathsf{dom}}} \to \mathcal{R}_q$$

and sets the equation that relates the parent node $\boldsymbol{p} \in \mathcal{R}_q^{\ell_{\mathsf{dom}}}$ with its children $\boldsymbol{c}_1, \boldsymbol{c}_2 \in \mathcal{R}_q^{\ell_{\mathsf{dom}}}$ as

$$h_{\mathsf{Ajtai}}(\boldsymbol{c}_1, \boldsymbol{c}_2) = \mathsf{proj}(\boldsymbol{p}) \mod q \ . \tag{1}$$

One may view $\boldsymbol{p}$ as some kind of encoding of $\mathsf{proj}(\boldsymbol{p})$ here. Since $\boldsymbol{p}$ enters the hash function on the next layer of the tree as a child, it must be small (this is checked by the verification algorithms along with the linear relation above). So to construct the tree, we need to be able to find small preimages of proj. In lattice terms, this means we need to solve some close(st) vector problem for the kernel of proj. An important observation is that this construction actually works for any proj for which we can find short preimages: the homomorphic properties of the HVC are due to the fact that equation (1) above is phrased in terms of $\mathcal{R}_q$-linear maps, and the sum of small elements stays small. We emphasize that what primarily matters here is the linearity properties of proj and the *verification* equation. The map that finds the small preimage may be thought of as auxilliary and will not be linear.

Squirrel chooses proj as binary reconstruction $\mathsf{proj}(\boldsymbol{p}) = p_0 + 2p_1 + 4p_2 + \ldots$. An algorithm to find a short inverse is then given by binary decomposition. To commit to the correct domain $\mathcal{R}_{q'}$, Squirrel adds some extra layers on the bottom of the tree.

The main improvement from Chipmunk over Squirrel comes from choosing a different map for proj and its inverse: we propose to instead use $(2\eta + 1)$-ary decomposition rather than binary decomposition. This turns out to give significantly better parameters.

Some other differences in the actual construction are as follows:

- We define the commitment (corresponding to the root of the Merkle tree) to be in non-decomposed form.
- We define proj and the $2\eta + 1$-adic decomposition as maps over $\mathcal{R}$ rather than $\mathcal{R}_q$.

3.2 A Homomorphic Vector Commitment based on Ring-SIS

To construct a homomorphic vector commitment with the desired properties, we will define proj and an inverse, called decomposition, as described above. We use $(2\eta+1)$-ary decomposition for the latter, which allows us to map a ring element with possibly large norm to a vector of low norm ring elements. To be able to use the greatest arity while minimizing the infinity norm of decomposed elements, we use a *balanced* $(2\eta + 1)$-ary decomposition, i.e. the decomposed elements have coefficients from $\{-\eta, \ldots, +\eta\}$ centered around 0. We note that any even arity, such as the binary decomposition used by Squirrel [13], is strictly worse than the next greater odd arity. We then show that the projection function has nice homomorphic properties.

Definition 3.6 (Projection onto $\mathcal{R}$ elements). Let $\eta, \kappa \in \mathbb{N}$. For any $\boldsymbol{b} \in \mathbb{Z}^\kappa$ we define the function

$$\mathsf{proj}_{\eta,\kappa} \colon \mathbb{Z}^\kappa \to \mathbb{Z}, \quad \mathsf{proj}_{\eta,\kappa}(\boldsymbol{b}) = \sum_{j=1}^{\kappa} b_j \cdot (2\eta + 1)^{j-1} \ .$$

We can extend this to a map

$$\mathsf{proj}_{\eta,\kappa}\colon \mathcal{R}^\kappa \to \mathcal{R}, \quad \mathsf{proj}_{\eta,\kappa}(\boldsymbol{b}) = \sum_{j=1}^{\kappa} b_j \cdot (2\eta+1)^{j-1} \ .$$

Definition 3.7 (Balanced $(2\eta+1)$-ary decomposition of $\mathcal{R}$ elements). Fix some odd arity $2\eta+1$ and let $\kappa \in \mathbb{N}$ be the number of limbs. Then we can uniquely decompose any $a \in \mathbb{Z}$ into a balanced $(2\eta+1)$-ary decomposition with κ limbs as

$$a = \sum_{i=1}^{\kappa} a_i \cdot (2\eta+1)^{i-1}$$

where $a_i \in \{-\eta, \dots, \eta\}$ for all $1 \le i < \kappa$. An algorithm and proof of this statement is given below in Figure 1 and Theorem 3.10. Note that it is notationally convenient to allow arbitrarily sized a in the definition and not bound a_κ, thereby putting all higher-order terms into a_κ. If we have the bound $|a| < \frac{(2\eta+1)^\kappa}{2}$, then the most significant limb a_κ will also be in $\{-\eta, \dots, \eta\}$.

We can extend this to a map on $\mathcal{R}$ by essentially decomposing each coefficient, uniquely mapping a polynomial $a \in \mathcal{R}$ to limbs $a_1 \dots, a_\kappa \in \mathcal{R}$ such that

$$a = \sum_{i=1}^{\kappa} a_i \cdot (2\eta+1)^{i-1}$$

where $\|a_i\|_\infty \le \eta$ for all $1 \le i < \kappa$. If $\|a\|_\infty < \frac{(2\eta+1)^\kappa}{2}$, we also have the bound $\|a_\kappa\|_\infty \le \eta$ for the most significant limb.

Matching the notation from Definition 3.6, we denote this decomposition map by $\mathsf{dec}_{\eta,\kappa}$, giving a map

$$\mathsf{dec}_{\eta,\kappa}\colon \mathcal{R} \to \mathcal{R}^\kappa, \quad a \mapsto (a_1, \dots, a_\kappa) \ .$$

Definition 3.8 (Projection and Decomposition for $\mathcal{R}_q$). Fix some odd arity $(2\eta+1)$ and let q be prime. Set $\kappa := \lceil \log_{2\eta+1} q \rceil$. We denote

$$\mathsf{proj}_q\colon \mathcal{R}^\kappa \to \mathcal{R}_q, \quad \mathsf{proj}_q(\boldsymbol{a}) := \mathsf{proj}_{\eta,\kappa}(\boldsymbol{a}) \bmod q \in \mathcal{R}_q$$

and

$$\mathsf{dec}_q\colon \mathcal{R}_q \to \mathcal{R}^\kappa, \quad \mathsf{dec}_q(a) := \mathsf{dec}_{\eta,\kappa}(a') \ ,$$

where a' is the representative of a in $\mathcal{R}$ with the coefficients being in $\{-\frac{q-1}{2}, \dots, +\frac{q-1}{2}\}$.

We remark that the only difference between proj_q and $\mathsf{proj}_{\eta,\kappa}$ resp. between dec_q and $\mathsf{dec}_{\eta,\kappa}$ is whether the non-decomposed element is in $\mathcal{R}_q$ or $\mathcal{R}$. The decomposed elements are always from $\mathcal{R}^\kappa$. For proj_q and dec_q, the value of η is not denoted explicitly. This is done for notational consistency with Squirrel. In our constructions, all uses of proj_q and dec_q will use the same value for η, even if the values of q differ.

The following proposition immediately follow from the definitions (for $\mathcal{R}$-linearity, this follows from the examples given after Definition 2.3).

Proposition 3.9. *Let q be an odd integer and fix some odd arity $2\eta+1$. The maps proj_q and $\mathsf{proj}_{\eta,\kappa}$ defined above are $\mathcal{R}$-linear. The map $\mathsf{dec}_{\eta,\kappa}$ is a one-sided inverse to $\mathsf{proj}_{\eta,\kappa}$, meaning that*

$$\mathsf{proj}_{\eta,\kappa}(\mathsf{dec}_{\eta,\kappa}(a)) = a$$

for any $a \in \mathcal{R}$. Similarly, dec_q is a one-sided inverse to proj_q, meaning that $\mathsf{proj}_q(\mathsf{dec}_q(a)) = a$ for any $a \in \mathcal{R}_q$. For $a \in \mathcal{R}_q$, we also have $\left\|\mathsf{dec}_q(a)\right\|_\infty \le \eta$.

$\mathsf{dec}_{\eta,\kappa}(a)$

$r_1 := a$
for $1 \le i \le \kappa - 1$
  Choose $a_i \in \{-\eta, \dots, +\eta\}$ with $a_i \equiv r_i \bmod (2\eta+1)$
  $r_{i+1} := \frac{r_i - a_i}{2\eta+1}$ / Numerator is divisible by $2\eta+1$
$a_\kappa := r_\kappa$
return $(a_1, \dots, a_\kappa)$

Figure 1: Algorithm for balanced $(2\eta+1)$-ary decomposition of integers $a \in \mathbb{Z}$. The corresponding algorithm for $a \in \mathcal{R}$ works by applying this coefficient-wise.

For the sake of readability we will at times abuse notation slightly and apply dec_q resp. $\mathsf{dec}_{\eta,\kappa}$ to *vectors* of $\mathcal{R}_q$ resp. $\mathcal{R}$ elements, which is to be understood as the component-wise application of dec_q resp. $\mathsf{dec}_{\eta,\kappa}$ with subsequent concatenation of the resulting vectors. Similarly, proj_q resp. $\mathsf{proj}_{\eta,\kappa}$ may be applied to vectors of a length that is a *multiple* of κ to result in a vector of $\mathcal{R}_q$ resp. $\mathcal{R}$ elements. The above discussion generalizes to this extension.

Proposition 3.10 (balanced $(2\eta+1)$-ary decomposition). *Let $\eta, \kappa \in \mathbb{N}$. The algorithm in Figure 1 runs in polynomial time. For any $a \in \mathbb{Z}$, it outputs the unique $(a_1, \dots, a_\kappa)$ with*

$$a = \sum_{i=1}^{\kappa} a_i \cdot (2\eta+1)^{i-1} \tag{2}$$

and $a_i \in \{-\eta, \dots, +\eta\}$ for $1 \le i \le \kappa - 1$.

Proof. The algorithm is clearly polynomial time. $r - a_i$ is divisible by $2\eta+1$ by construction of a_i. By definition, $a_1, \dots, a_{\kappa-1} \in \{-\eta, \dots, +\eta\}$. We show by induction that we have for all $1 \le i \le \kappa - 1$

$$a = r_i \cdot (2\eta+1)^{i-1} + \sum_{j=1}^{i-1} a_j (2\eta+1)^{j-1} \ .$$

This is clear for $i = 1$. Using induction, we compute

$$\begin{aligned}
& r_{i+1} \cdot (2\eta+1)^i + \sum_{j=1}^{i} a_j (2\eta+1)^{j-1} \\
=& (r_i - a_i) \cdot (2\eta+1)^{i-1} + \sum_{j=1}^{i} a_j (2\eta+1)^{j-1} && \text{(Def. of } r_{i+1}\text{)} \\
=& r_i \cdot (2\eta+1)^{i-1} + \sum_{j=1}^{i-1} a_j (2\eta+1)^{j-1} = a && \text{(induction hypothesis)}
\end{aligned}$$

For $i = \kappa$, this yields $a = \sum_{i=1}^{\kappa} a_i \cdot (2\eta+1)^{i-1}$. For uniqueness, note that we just showed that the map $\{-\eta, \dots, \eta\}^{\kappa-1} \times \mathbb{Z} \to \mathbb{Z}$, $(a_1, \dots, a_\kappa) \mapsto \sum_{i=1}^{\kappa} a_i \cdot (2\eta+1)^{i-1}$ is surjective. Taking this modulo $(2\eta+1)^{\kappa-1}$ gives us that $\{-\eta, \dots, +\eta\}^{\kappa-1} \to \mathbb{Z}_{(2\eta+1)^{\kappa-1}}$, $(a_1, \dots, a_{\kappa-1}) \mapsto \sum_{i=1}^{\kappa-1} a_i \cdot (2\eta+1)^{i-1} \bmod (2\eta+1)^{\kappa-1}$ is surjective, hence injective (because domain and range have the same finite size). So $a_1, \dots, a_{\kappa-1}$ are uniquely determined. Plugging this into Equation 2 shows that a_κ is uniquely determined as well. □

This lets us define a labeling function for a full binary tree.

Definition 3.11 (Labeled Full Binary Tree). Let $n, q, q', \xi \in \mathbb{N}$ with n a power of two and q, q' primes. Let $\boldsymbol{m} = (\boldsymbol{m}_1, \ldots, \boldsymbol{m}_{2^\tau})^\intercal \in (\mathcal{R}_{q'}^{\xi})^{2^\tau}$, $\boldsymbol{g} \in \mathcal{R}_q^{\xi\kappa'}$ and $\boldsymbol{h}_0, \boldsymbol{h}_1 \in \mathcal{R}_q^{\kappa}$ be fixed. We define the labeling function $\mathsf{label}_{\boldsymbol{g},\boldsymbol{h}_0,\boldsymbol{h}_1} \colon (\mathcal{R}_{q'}^{\xi})^{2^\tau} \times \{0,1\}^{\leq\tau} \to \mathcal{R}^{\lceil \log_{2\eta+1} q \rceil}$ for a labeled full binary tree of depth τ as

$$\mathsf{label}_{\boldsymbol{g},\boldsymbol{h}_0,\boldsymbol{h}_1}(\boldsymbol{m}, v) := \begin{cases} \mathsf{dec}_q(\boldsymbol{g}^\intercal \cdot \mathsf{dec}_{q'}(\boldsymbol{m}_{v+1})) & \text{if } |v| = \tau \\ \mathsf{dec}_q\left(\begin{matrix} \boldsymbol{h}_0^\intercal \cdot \mathsf{label}_{\boldsymbol{g},\boldsymbol{h}_0,\boldsymbol{h}_1}(\boldsymbol{m}, v\|0) \\ +\boldsymbol{h}_1^\intercal \cdot \mathsf{label}_{\boldsymbol{g},\boldsymbol{h}_0,\boldsymbol{h}_1}(\boldsymbol{m}, v\|1) \end{matrix}\right) & \text{if } |v| < \tau \end{cases}$$

For this, remember that multiplication of elements from $\mathcal{R}_q$ and $\mathcal{R}$ is always understood[5] to give an element in $\mathcal{R}_q$. For $\boldsymbol{m}_{v+1}$, we interpret $v \in \{0,1\}^\tau$ as an integer in $\{0 \ldots, 2^\tau - 1\}$ in big-endian encoding and add 1 (i.e. the inverse of taking $\tilde{t} = \mathsf{bin}_\tau(t-1)$).

Using the labeling function, we can define Chipmunk's HVC as in Figure 2.

Definition 3.12. Let $n, q, q', \alpha_w, \rho, \eta, \tau, \xi, \beta_{\mathsf{agg}}$ be positive integers such that n is a power of two and q, q' are primes. Let $\mathcal{R}_q, \mathcal{R}_{q'}$ be the polynomial rings $\mathbb{Z}_q[X]/\langle X^n+1\rangle$ and $\mathbb{Z}_{q'}[X]/\langle X^n+1\rangle$ respectively. We define the homomorphic vector commitment $\mathsf{HVC}^{\mathsf{Chip}}$ for domain $A_{\mathsf{dom}} = \mathcal{R}_{q'}^{\xi}$ and vectors of length 2^τ by the algorithms given in Figure 2. Its commitments and openings are from $A_{\mathsf{com}} = \mathcal{R}_q$ and $A_{\mathsf{op}} = (\mathcal{R}^\kappa)^{2\tau} \times (\mathcal{R}^{\kappa'})^{\xi}$, where $\kappa = \lceil \log_{2\eta+1} q \rceil, \kappa' = \lceil \log_{2\eta+1} q' \rceil$.

Remark 3.13. Before proving security of $\mathsf{HVC}^{\mathsf{Chip}}$, let us give some remarks on the construction itself.

(1) The tree labels constructed by the labeling function that constitute the Merkle path $\boldsymbol{p}_j$ with its sibling nodes $\boldsymbol{s}_j$ are *decomposed* elements, i.e. short elements in $\mathcal{R}$. For efficiency reasons, the commitment $\boldsymbol{c}$ itself is not $\boldsymbol{p}_0$, but rather in non-decomposed form. This is done to ensure the commitment is in $\mathcal{R}_q$ rather than $\mathcal{R}$, which is slightly more efficient when aggregating. Regarding analysis, observe that if we set $\boldsymbol{p}_0$ as in the definition of Com, the condition $\boldsymbol{c} = \boldsymbol{h}_{\tilde{t}_1}^\intercal \cdot \boldsymbol{p}_1 + \boldsymbol{h}_{\tilde{t}_1 \oplus 1}^\intercal \cdot \boldsymbol{s}_1$ is actually equivalent to
$$\mathsf{proj}_q(\boldsymbol{p}_0) = \boldsymbol{h}_{\tilde{t}_1}^\intercal \cdot \boldsymbol{p}_1 + \boldsymbol{h}_{\tilde{t}_1 \oplus 1}^\intercal \cdot \boldsymbol{s}_1 \ .$$
Hence, we may treat this condition as the special case $j = 1$ of the condition $\mathsf{proj}_q(\boldsymbol{p}_{j-1}) = \boldsymbol{h}_{\tilde{t}_j}^\intercal \cdot \boldsymbol{p}_j + \boldsymbol{h}_{\tilde{t}_j \oplus 1}^\intercal \cdot \boldsymbol{s}_j$.

(2) The full version of this work [12] contains a encoding scheme to transmit decommitments more efficiently. This requires some extra bounds checks on $\mathsf{proj}_{\eta,\kappa}(\boldsymbol{p}_j)$ and $\mathsf{proj}_{\eta,\kappa}(\boldsymbol{s}_j)$ inside Vf, which are not present here.

Theorem 3.14. *Let $n, q, q', \alpha_w, \rho, \eta, \tau, \xi, \beta_{\mathsf{agg}}$ be positive integers and $0 < \varepsilon \leq 1$ such that n is a power of two, q, q' are prime, and*

$$\beta_{\mathsf{agg}} \geq \eta\sqrt{2\alpha_w\rho\left(\ln\tfrac{2n}{\varepsilon} + \ln(2\tau\kappa + \xi\kappa')\right)} \ .$$

where $\kappa = \lceil \log_{2\eta+1} q \rceil$ and $\kappa' = \lceil \log_{2\eta+1} q' \rceil$. Let $\mathcal{R}_q, \mathcal{R}_{q'}$ be the polynomial rings $\mathbb{Z}_q[X]/\langle X^n+1\rangle$ and $\mathbb{Z}_{q'}[X]/\langle X^n+1\rangle$ respectively. If the problems $\mathsf{SIS}_{\mathcal{R},q,2\kappa q,4\beta_{\mathsf{agg}}}$ and $\mathsf{SIS}_{\mathcal{R},q,\xi\kappa',4\beta_{\mathsf{agg}}}$ are hard, then $\mathsf{HVC}^{\mathsf{Chip}}$ is an individually correct, a $(\rho, \mathcal{T}_{\alpha_w}, \varepsilon)$-probabilistically homomorphic, robustly homomorphic, and position binding HVC for domain $\mathcal{R}_{q'}^{\xi}$ and vector length 2^τ.

Due to space constraints the proof of Theorem 3.14 is ommitted. The proof follows the same ideas as the proof in [13] and can be found in the full version [12] of this work.

[5] as opposed to taking some canonical representative of $\mathcal{R}_q$-elements in $\mathcal{R}$ and then multiplying in $\mathcal{R}$

4 KEY-HOMOMORPHIC ONE-TIME SIGNATURES

In this section, we define and instantiate key-homomorphic one-time signatures, which are a weak form of a digital signature scheme that is only guaranteed to be unforgeable, if at most one signature is published under any given public key. A one-time signature is called key-homomorphic, if the linear combination of separate signatures for the same message verifies under the linear combination of the corresponding public keys.

Our definitions again follow the definitions of [13] closely, but are incomparable just as in Section 3. As with the vector commitments, we have the stronger requirement that the homomorphism works for any individually verifying signature, not just honestly created ones. But, this homomorphism is allowed to have a noticeable correctness error.

The construction presented in this section is a modification of the construction of [13], which itself was a modification of the one-time signature schemes by Boneh and Kim [6] and Lyubashevsky and Micciancio [21].

Definition 4.1 (Key-Homomorphic One-Time Signature). Let $\mathcal{R}$ be a ring. Let A_{opk} and A_{sig} be $\mathcal{R}$-modules denoting the spaces where the public keys and signatures are from. A key-homomorphic one-time signature scheme (KOTS) over $\mathcal{R}$ with public key space A_{opk} and signatures from A_{sig} is defined by six PPT algorithms $\mathsf{KOTS} = (\mathsf{Setup}, \mathsf{KG}, \mathsf{Sig}, \mathsf{iVf}, \mathsf{sVf}, \mathsf{wVf})$.

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ The setup algorithm takes as input the security parameter and outputs public parameters.
- $(\mathsf{osk}, \mathsf{opk}) \leftarrow \mathsf{KG}(\mathsf{pp})$ The key generation algorithm takes as input the public parameters and outputs a key pair with $\mathsf{opk} \in A_{\mathsf{opk}}$.
- $\sigma \leftarrow \mathsf{Sig}(\mathsf{pp}, \mathsf{osk}, m)$ The signing algorithm takes as input the public parameters, a one-time signing key, and a message and outputs a signature $\sigma \in A_{\mathsf{sig}}$.
- $b \leftarrow \mathsf{iVf}(\mathsf{pp}, \mathsf{opk}, m, \sigma)$ The individual verification algorithm takes as input the public parameters, a verification key, a message, and a candidate signature and outputs a bit indicating acceptance/rejection.
- $b \leftarrow \mathsf{sVf}(\mathsf{pp}, \mathsf{opk}, m, \sigma)$ The strong verification algorithm has the same input and output domains as the individual verification algorithm.
- $b \leftarrow \mathsf{wVf}(\mathsf{pp}, \mathsf{opk}, m, \sigma)$ The weak verification algorithm has the same input and output domains as the individual verification algorithm.

Note that for us, we will always have $\mathcal{R} = \mathbb{Z}[X]/\langle X^n+1\rangle$ for n a power of two and $A_{\mathsf{sig}} = \mathcal{R}_{q'}^{\ell_{\mathsf{sig}}}$, $A_{\mathsf{opk}} = \mathcal{R}_{q'}^{\ell_{\mathsf{opk}}}$ for some prime q'.

Definition 4.2 (Individual Correctness). Let KOTS be a key-homomorphic one-time signature scheme. KOTS is individually correct, if for all security parameters $\lambda \in \mathbb{N}$, parameters $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, key

$\underline{\mathsf{Setup}(1^\lambda)}$
$\boldsymbol{g} \leftarrow \mathcal{R}_q^{\xi\kappa'}$
$\boldsymbol{h}_0 \leftarrow \mathcal{R}_q^{\kappa}$
$\boldsymbol{h}_1 \leftarrow \mathcal{R}_q^{\kappa}$
return $(\boldsymbol{g}, \boldsymbol{h}_0, \boldsymbol{h}_1)$

$\underline{\mathsf{Com}(\mathsf{pp}, \boldsymbol{m})}$
$\boldsymbol{p}_0 := \mathsf{label}_{\boldsymbol{g},\boldsymbol{h}_0,\boldsymbol{h}_1}(\boldsymbol{m}, \epsilon)$
$\boldsymbol{c} := \mathsf{proj}_q(\boldsymbol{p}_0)$
return $\boldsymbol{c} \in \mathcal{R}_q$

$\underline{\mathsf{Open}(\mathsf{pp}, \boldsymbol{c}, \boldsymbol{m}, t)}$
$\tilde{t} := \mathsf{bin}_\tau(t-1)$
for $1 \le j \le \tau$
  $\boldsymbol{p}_j := \mathsf{label}_{\boldsymbol{g},\boldsymbol{h}_0,\boldsymbol{h}_1}(\boldsymbol{m}, \tilde{t}_{<j} \| \tilde{t}_j)$
  $\boldsymbol{s}_j := \mathsf{label}_{\boldsymbol{g},\boldsymbol{h}_0,\boldsymbol{h}_1}(\boldsymbol{m}, \tilde{t}_{<j} \| (\tilde{t}_j \oplus 1))$
$\boldsymbol{u} := \mathsf{dec}_{q'}(\boldsymbol{m}_t)$
return $(\boldsymbol{p}_1, \ldots, \boldsymbol{p}_\tau, \boldsymbol{s}_1, \ldots, \boldsymbol{s}_\tau, \boldsymbol{u})$

$\underline{\mathsf{Vf}(\mathsf{pp}, \boldsymbol{c}, t, \boldsymbol{d}, \beta)}$
parse $\boldsymbol{d}$ **as** $(\boldsymbol{p}_1, \ldots, \boldsymbol{p}_\tau, \boldsymbol{s}_1, \ldots, \boldsymbol{s}_\tau, \boldsymbol{u})$
$\tilde{t} := \mathsf{bin}_\tau(t-1)$
if $\|\boldsymbol{u}\| > \beta$ **or** $\boldsymbol{g}^\intercal \cdot \boldsymbol{u} \neq \mathsf{proj}_q(\boldsymbol{p}_\tau)$
  return $\perp$
if $\boldsymbol{c} \neq \boldsymbol{h}_{\tilde{t}_1}^\intercal \cdot \boldsymbol{p}_1 + \boldsymbol{h}_{\tilde{t}_1 \oplus 1}^\intercal \cdot \boldsymbol{s}_1$
  return $\perp$
for $2 \le j \le \tau$
  if $\mathsf{proj}_q(\boldsymbol{p}_{j-1}) \neq \boldsymbol{h}_{\tilde{t}_j}^\intercal \cdot \boldsymbol{p}_j + \boldsymbol{h}_{\tilde{t}_j \oplus 1}^\intercal \cdot \boldsymbol{s}_j$
    return $\perp$
for $j \in \{1, \ldots, \tau\}$
  if $\|\boldsymbol{p}_j\| > \beta$ **or** $\|\boldsymbol{s}_j\| > \beta$
    return $\perp$
return $\mathsf{proj}_{q'}(\boldsymbol{u}) \in \mathcal{R}_{q'}^{\xi}$

$\underline{\mathsf{iVf}(\mathsf{pp}, \boldsymbol{c}, t, \boldsymbol{d})}$
return $\mathsf{Vf}(\mathsf{pp}, \boldsymbol{c}, t, \boldsymbol{d}, \eta)$

$\underline{\mathsf{sVf}(\mathsf{pp}, \boldsymbol{c}, t, \boldsymbol{d})}$
return $\mathsf{Vf}(\mathsf{pp}, \boldsymbol{c}, t, \boldsymbol{d}, \beta_{\mathrm{agg}})$

$\underline{\mathsf{wVf}(\mathsf{pp}, \boldsymbol{c}, t, \boldsymbol{d})}$
return $\mathsf{Vf}(\mathsf{pp}, \boldsymbol{c}, t, \boldsymbol{d}, 2\beta_{\mathrm{agg}})$

Figure 2: The construction of the homomorphic vector commitment $\mathsf{HVC}^{\mathbf{Chip}}$ for message space $A_{\mathsf{dom}} = \mathcal{R}_{q'}^{\xi}$ based on a labeled binary tree, cf. Definition 3.12. Commitments $\boldsymbol{c}$ are in $A_{\mathsf{com}} = \mathcal{R}_q$. Openings are small elements in $A_{\mathsf{op}} = (\mathcal{R}^\kappa)^{2\tau} \times (\mathcal{R}^{\kappa'})^{\xi}$, where $\kappa = \lceil \log_{2\eta+1} q \rceil$, $\kappa' = \lceil \log_{2\eta+1} q' \rceil$. Note that multiplication of $\mathcal{R}_q$ with $\mathcal{R}$ elements as in the computation of Ajtai hashes is understood to give an element in $\mathcal{R}_q$, i.e. modular reduction is performed.

pairs $(\mathsf{osk}, \mathsf{opk}) \leftarrow \mathsf{KG}(\mathsf{pp})$, messages $m \in \{0,1\}^*$, and signatures $\boldsymbol{\sigma} \leftarrow \mathsf{Sig}(\mathsf{pp}, \mathsf{osk}, m)$ it holds that

$$\mathsf{iVf}(\mathsf{pp}, \mathsf{opk}, m, \sigma) = 1 \ .$$

We require that individually verifying signatures can be homomorphically aggregated by computing a random linear combination of them. Such aggregated signatures should still *strongly* verify with high probability over the choice of the random linear combination.

Definition 4.3 (Probabilistic Homomorphism). Let KOTS be a one-time signature scheme over a ring $\mathcal{R}$ with public key space A_{opk} and signatures from A_{sig}. Let $\rho \in \mathbb{N}$, error bound $0 \le \varepsilon \le 1$ and $W \subseteq \mathcal{R}$. KOTS is (ρ, W, ε)-probabilistically homomorphic, if for all security parameters $\lambda \in \mathbb{N}$, number of aggregated signatures $\ell \in [\rho]$, parameters $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, public keys $\mathsf{opk}^i \in A_{\mathsf{opk}}$, messages $m \in \{0,1\}^*$ and signatures $\boldsymbol{\sigma}^i \in A_{\mathsf{sig}}$ with $\mathsf{iVf}(\mathsf{pp}, \mathsf{opk}^i, m, \boldsymbol{\sigma}^i)$ it holds that

$$\Pr\left[w^1, \ldots, w^\ell \leftarrow W : \mathsf{sVf}(\mathsf{pp}, \sum_{i=1}^{\ell} w^i \cdot \mathsf{opk}^i, m, \sum_{i=1}^{\ell} w^i \cdot \boldsymbol{\sigma}^i) = 1 \right] \ge 1 - \varepsilon \ .$$

As with the vector commitments from the previous section, we additionally require that a further limited homomorphism still holds, even for maliciously *aggregated* signatures. For any two, even maliciously generated, signatures that *strongly* verify under potentially maliciously generated public keys, their difference will still *weakly* verify.

Definition 4.4 (Robust Homomorphism). Let KOTS be a key-homomorphic one-time signature scheme over a ring $\mathcal{R}$ with public key space A_{opk} and signatures from A_{sig}. KOTS is robustly homomorphic if for all security parameters $\lambda \in \mathbb{N}$, public parameters $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, messages $m \in \{0,1\}^*$, (possibly malformed) public keys $\mathsf{opk}^0, \mathsf{opk}^1 \in A_{\mathsf{opk}}$, and (possibly malformed) signatures $\boldsymbol{\sigma}^0, \boldsymbol{\sigma}^1 \in A_{\mathsf{sig}}$ with

$$\mathsf{sVf}(\mathsf{pp}, \mathsf{opk}^0, m, \boldsymbol{\sigma}^0) = 1 \quad \text{and} \quad \mathsf{sVf}(\mathsf{pp}, \mathsf{opk}^1, m, \boldsymbol{\sigma}^1) = 1$$

it holds that $\mathsf{wVf}(\mathsf{pp}, \mathsf{opk}^0 - \mathsf{opk}^1, m, (\boldsymbol{\sigma}^0 - \boldsymbol{\sigma}^1)) = 1$.

The following definition of a multi-user version of (one-time) existential unforgeability under rerandomized keys is taken directly from [13].

Definition 4.5 (Multi-User Existential Unforgeability under Rerandomized Keys). A (ρ, W, ε)-homomorphically correct KOTS is W'-existentially unforgeable under rerandomized keys (EUF-RK), if for all security parameters λ, any $T = \mathsf{poly}(\lambda) \in \mathbb{N}$ and all stateful PPT algorithms $\mathcal{A}$ it holds that

$$\Pr\left[\begin{array}{r} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda); \\ \forall i \in [T].\ (\mathsf{osk}_i, \mathsf{opk}_i) \leftarrow \mathsf{KG}(\mathsf{pp}); \\ \begin{pmatrix} i^*, m^* \\ \sigma^*, w^* \end{pmatrix} \leftarrow \mathcal{A}^{\widetilde{\mathsf{Sig}}(\cdot,\cdot)}(\mathsf{pp}, \mathbf{opk}) \end{array} : \begin{array}{l} \mathsf{wVf}(\mathsf{pp}, w^* \mathsf{opk}_{i^*}, m^*, \sigma^*) = 1 \\ \wedge\ m^* \notin Q_{i^*} \wedge |Q_{i^*}| \le 1 \\ \wedge\ w^* \in W' \end{array} \right] \le \mathsf{negl}(\lambda) \ ,$$

where the oracle $\widetilde{\mathsf{Sig}}(\cdot,\cdot)$ is defined as $\widetilde{\mathsf{Sig}}(i, m) := \mathsf{Sig}(\mathsf{osk}_i, m)$ and Q_i denotes the set of messages for which a signing query with index i has been made.

Figure 3 shows the construction of the KOTS we will use. The construction is almost identical to the construction from [13] but differs from it in its choice of the ball from which the secret keys are chosen. Specifically, the components of the secret keys are allowed to have a larger infinity norm. This is beneficial, because the security proof partially relies on fact that the function mapping secret keys to public keys and signatures is highly compressing. With a larger secret key-space the compression ratio increases, allowing us to reduce the size of other parameters, ultimately decreasing the size of the signatures.

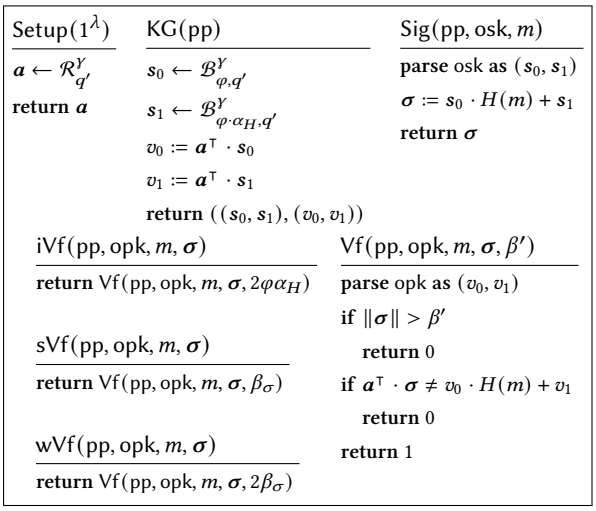
Setup(1^λ)
$\boldsymbol{a} \leftarrow \mathcal{R}^Y_{q'}$
return $\boldsymbol{a}$
KG(pp)
$s_0 \leftarrow \mathcal{B}^Y_{\varphi,q'}$
$s_1 \leftarrow \mathcal{B}^Y_{\varphi\cdot\alpha_H,q'}$
$v_0 := \boldsymbol{a}^\top \cdot s_0$
$v_1 := \boldsymbol{a}^\top \cdot s_1$
return $((s_0, s_1), (v_0, v_1))$
Sig(pp, osk, m)
parse osk as (s_0, s_1)
$\boldsymbol{\sigma} := s_0 \cdot H(m) + s_1$
return $\boldsymbol{\sigma}$
iVf(pp, opk, m, $\boldsymbol{\sigma}$)
return Vf(pp, opk, m, $\boldsymbol{\sigma}$, $2\varphi\alpha_H$)
sVf(pp, opk, m, $\boldsymbol{\sigma}$)
return Vf(pp, opk, m, $\boldsymbol{\sigma}$, β_σ)
wVf(pp, opk, m, $\boldsymbol{\sigma}$)
return Vf(pp, opk, m, $\boldsymbol{\sigma}$, $2\beta_\sigma$)
Vf(pp, opk, m, $\boldsymbol{\sigma}$, β')
parse opk as (v_0, v_1)
if $\|\boldsymbol{\sigma}\| > \beta'$
return 0
if $\boldsymbol{a}^\top \cdot \boldsymbol{\sigma} \neq v_0 \cdot H(m) + v_1$
return 0
return 1

Figure 3: Description of our key-homomorphic one-time signature scheme $\mathsf{KOTS}^{\mathsf{Chip}}$ from Definition 4.6. H is a collision-resistant hash function mapping bit-strings to $\mathcal{T}_{\alpha_H}$. Our key space is $A_{\mathsf{opk}} = \mathcal{R}^2_{q'}$. The signature space is $A_{\mathsf{sig}} = \mathcal{R}^Y_{q'}$.

Definition 4.6. Let $n, q', \alpha_H, \varphi, \gamma, \beta_\sigma$ be integers and H be a hash function mapping bit strings to $\mathcal{T}_{\alpha_H}$. Let $\mathcal{R} = \mathbb{Z}[X]/\langle X^n + 1\rangle$ and $\mathcal{R}_{q'} = \mathbb{Z}_{q'}[X]/\langle X^n + 1\rangle$. We define $\mathsf{KOTS}^{\mathsf{Chip}} = (\mathsf{Setup}, \mathsf{KG}, \mathsf{Sig}, \mathsf{iVf}, \mathsf{sVf}, \mathsf{wVf})$ as the key-homomorphic one-time scheme over $\mathcal{R}$ as in Figure 3. Its public key space is $A_{\mathsf{opk}} = \mathcal{R}^2_{q'}$ and its signature space is $A_{\mathsf{sig}} = \mathcal{R}^Y_{q'}$.

Theorem 4.7. *Let $\lambda, \alpha_w, \alpha_H, \varphi, \gamma, \rho, \beta_\sigma, n, q'$ be integers and $0 < \varepsilon < 1$ such that q' is prime and $q' > 16\alpha_w\alpha_H\varphi$, n is a power of two and there exists δ with*

$$2^{2\lambda} \leq \left|\mathcal{T}_{\alpha_H}\right| \leq 2^{2\lambda+\delta},$$
$$\beta_\sigma \geq 4\varphi\alpha_H\sqrt{\tfrac{1}{2}\alpha_w\rho \cdot \ln\tfrac{2n\gamma}{\varepsilon}}$$
$$\gamma \geq ((3\lambda+\delta)/n + \log_2 q')\log_2^{-1}(\varphi + \tfrac{1}{2}) \ .$$

Let $H\colon \{0,1\}^* \to \mathcal{T}_{\alpha_H}$ be a hash function. Let $W' = \{w_0 - w_1 \mid w_0, w_1 \in \mathcal{T}_{\alpha_w} \wedge w_0 \neq w_1\}$. If the $\mathsf{SIS}_{\mathcal{R},q',\gamma,2\beta_\sigma+4\alpha_w\alpha_H\varphi}$ problem is hard and H is collision resistant, then the construction $\mathsf{KOTS}^{\mathsf{Chip}}$ from Figure 3 is an individually correct, $(\rho, \mathcal{T}_{\alpha_w}, \varepsilon)$-probabilistically homomorphic, robustly homomorphic KOTS that is W'-multi-user existentially unforgeable under rerandomized keys.

Due to space constraints the proof of Theorem 4.7 is ommitted. The proof follows the same ideas as the proof in [13] with the neccessary modifications. The proof can be found in the full version [12] of this work.

5 SYNCHRONIZED MULTI-SIGNATURES

In this section, we show how the tools developed in the previous sections can be combined to yield a synchronized multi-signature with the desired properties. We do this in a manner that is almost identical to the way Squirrel [13] does it, except that our aggregation is modified to use a rejection sampling technique that allows us to reduce the signature size. Roughly speaking, a public key in the multi-signature scheme is a vector commitment to a vector of independent one-time signature public keys. To sign a message at time t, the signer publishes an opening to the key in vector position t and signs the message with that key.

To aggregate these signatures, the construction computes a random linear combination of them, using weights derived using a random oracle. The uniform distribution of weights allows us to leverage the probabilistic homomorphism of the KOTS and HVC schemes, such that this aggregation procedure will be successful with probability at least $1 - 2\varepsilon$. By rejecting unsuccessful attempts and retrying a number of times, the overall probability of an aggregation failure can be made negligible.

We will now formally define the requirements for a synchronized multi-signature scheme. Once again, our definitions follow the definitions of Fleischhacker, Simkin, and Zhang [13]. In contrast to their work, however, we define a significantly stronger notion of correctness for aggregated signatures. More concretely, [13] only required that aggregation is successful for honestly generated keys and signatures. We, on the other hand, require that any sequence of *individually valid* signatures can be successfully aggregated.[6]

Definition 5.1 (Synchronized Multi-Signatures). A synchronized ρ-wise multi-signature scheme for 2^τ time periods is defined by six PPT algorithms $\mathsf{MSIG} = (\mathsf{Setup}, \mathsf{KG}, \mathsf{Sig}, \mathsf{Aggregate}, \mathsf{iVf}, \mathsf{aVf})$.

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ The setup algorithm takes as input the security parameter and outputs public parameters pp.
- $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KG}(\mathsf{pp})$ The key generation algorithm takes as input the public parameters and outputs a key-pair.
- $\sigma \leftarrow \mathsf{Sig}(\mathsf{pp}, \mathsf{sk}, t, m)$ The signing algorithm takes as input the public parameters, a secret key, a time period $1 \leq t \leq 2^\tau$, and a message and outputs a signature.
- $\sigma_{\mathsf{agg}} \leftarrow \mathsf{Aggregate}(\mathsf{pp}, \mathcal{P}, t, m, \mathcal{S})$ The aggregation algorithm takes as input the public parameters, a list of public keys, a time period $1 \leq t \leq 2^\tau$, a message, and a list of signatures, where $|\mathcal{P}| = |\mathcal{S}| \leq \rho$ and outputs an aggregated signature or an error $\perp$.
- $b \leftarrow \mathsf{iVf}(\mathsf{pp}, \mathsf{pk}, t, m, \sigma)$ The deterministic individual verification algorithm takes as input the public parameters, a public key, a time period $1 \leq t \leq 2^\tau$, a message, and a signature and outputs a bit indicating acceptance/rejection.
- $b \leftarrow \mathsf{aVf}(\mathsf{pp}, \mathcal{P}, t, m, \sigma_{\mathsf{agg}})$ The deterministic aggregated verification algorithm takes as input the public parameters, a list of public keys, a time period $1 \leq t \leq 2^\tau$, a message, and an aggregated signature and outputs a bit indicating acceptance/rejection.

Definition 5.2 (Individual Correctness). Let MSIG be a synchronized ρ-wise multi-signature scheme for 2^τ time periods. MSIG is individually correct if for all security parameters $\lambda \in \mathbb{N}$, public parameters $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, key pairs $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KG}(\mathsf{pp})$, time periods $1 \leq t \leq 2^\tau$, message $m \in \{0,1\}^*$, and signatures $\sigma \leftarrow \mathsf{Sig}(\mathsf{pp}, \mathsf{sk}, t, m)$ it holds that

$$\mathsf{iVf}(\mathsf{pp}, \mathsf{pk}, t, m, \sigma) = 1 \ .$$

[6] It is worth noting, that the *construction* of Squirrel [13] actually satisfies this stronger notion. It was just never defined or proven.

Definition 5.3 (Aggregation Correctness with Rogue Keys and Signatures). Let MSIG be a synchronized ρ-wise multi-signature scheme for 2^τ time periods. MSIG has correct aggregations in the presence of rogue keys and signatures if for all security parameters $\lambda \in \mathbb{N}$, public parameters $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, number of aggregated signatures $\ell \in [\rho]$, time periods $1 \le t \le 2^\tau$, messages $m \in \{0,1\}^*$, public keys $\mathcal{P} = (\mathsf{pk}^1, \ldots, \mathsf{pk}^\ell)$ and signatures $\mathcal{S} = (\sigma^1, \ldots, \sigma^\ell)$, such that for all $i \in [\ell]$, $\mathsf{iVf}(\mathsf{pp}, \mathsf{pk}^i, t, m, \sigma^i) = 1$ it holds that

$$\Pr[\sigma_{\mathsf{agg}} \leftarrow \mathsf{Aggregate}(\mathsf{pp}, \mathcal{P}, t, m, \mathcal{S}) \colon \mathsf{aVf}(\mathsf{pp}, \mathcal{P}, t, m, \sigma_{\mathsf{agg}}) = 1] = 1 - \mathsf{negl}(\lambda) \ .$$

Definition 5.4 (Unforgeability). Let MSIG be a synchronized ρ-wise multi-signature scheme for 2^τ time periods. MSIG is unforgeable if for all security parameters $\lambda \in \mathbb{N}$, and all PPT algorithms $\mathcal{A}$ it holds that

$$\Pr\left[\begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda); \\ (\mathsf{sk}^*, \mathsf{pk}^*) \leftarrow \mathsf{KG}(\mathsf{pp}); \\ \begin{pmatrix}\mathcal{P}, t, \\ m, \sigma_{\mathsf{agg}}\end{pmatrix} \leftarrow \mathcal{A}^{\mathsf{Sig}(\mathsf{sk}^*,\cdot,\cdot)}(\mathsf{pp}, \mathsf{pk}^*) \end{array} : \begin{array}{l} \begin{pmatrix}\mathsf{aVf}(\mathsf{pp}, \mathcal{P}, t, m, \sigma_{\mathsf{agg}}) \\ = 1\end{pmatrix} \\ \wedge\ \mathsf{pk}^* \in \mathcal{P} \\ \wedge \begin{pmatrix}\forall\ (t', m', \sigma') \in Q. \\ (t', m') \neq (t, m)\end{pmatrix} \\ \wedge\ \forall\ t'.\ |Q_{t'}| \le 1 \end{array}\right] \le \mathsf{negl}(\lambda) \ ,$$

where Q denotes the set of signing queries made by $\mathcal{A}$ and $Q_{t'}$ denotes the set of signing queries made for timeslot t'.

5.1 Construction

For ease of notation we define the function zip that "zips" up two vectors into a single vector of pairs, i.e. $\mathsf{zip}(\boldsymbol{a}, \boldsymbol{b}) := ((a_1, b_1), \ldots, (a_\ell, b_\ell))$. The following theorem now states the security of our construction presented in Figure 4.

Theorem 5.5. *Let $\lambda, n, q', \xi, \chi, \tau$ be positive integers and $0 < \varepsilon < \frac{1}{2}$ with n being a power of two, q' being prime, and $\chi \ge \lambda / \log_2(\frac{1}{2\varepsilon})$. Let $\mathcal{R}_{q'}$ be the polynomial ring $\mathbb{Z}_{q'}[X]/\langle X^n + 1\rangle$. Let $W \subseteq \mathcal{R}$ be a set such that $|W| > 2^\lambda$ and let $W' := \{w^0 - w^1 | w^0, w^1 \in W\}$. Let $H \colon \{0,1\}^* \to W^\rho$ be a random oracle. Let* KOTS *be a key homomorphic one-time signature scheme with public keys in $\mathcal{R}_{q'}^\xi$ and let* HVC *be a homomorphic vector commitment for domain $\mathcal{R}_{q'}^\xi$.*

If KOTS *is individually correct, (ρ, W, ε)-probabilistically homomorphic, robustly homomorphic, and W'-multi-user existentially unforgeable under rerandomized keys and* HVC *is individually correct, (ρ, W, ε)-probabilistically homomorphic, robustly homomorphic, and position-binding, then the construction from Figure 4 is an unforgeable synchronized ρ-wise multi-signature scheme that is individually correct and has correct aggregations in the presence of rogue keys and signatures.*

Due to space constraints the proof of Theorem 5.5 is ommitted. The proof follows the same ideas as the proof in [13], except that we also need to show that aggregation succeeds with high probability. The proof can be found in the full version [12] of this work.

Our concrete proposal is to use $\mathsf{HVC}^{\mathsf{Chip}}$ and $\mathsf{KOTS}^{\mathsf{Chip}}$ to thereby construct the Chipmunk synchronized multi-signature.

Definition 5.6 (Chipmunk synchronized multi-signatures). Let $n, q, q', \eta, \tau, \rho, \alpha_H, \alpha_w, \gamma$ be positive integers, with n being a power of two, q, q' prime. Let $\mathcal{R}, \mathcal{R}_q, \mathcal{R}_{q'}$ be as usual. We define the synchronized multi-signature Chipmunk, denoted $\mathsf{MSIG}^{\mathsf{Chip}}$, by instanciating the construction from Figure 4 with $\mathsf{HVC}^{\mathsf{Chip}}$ with $\xi = 2$ and $\mathsf{KOTS}^{\mathsf{Chip}}$.

As a corollary of Theorem 5.5, we obtain that $\mathsf{MSIG}^{\mathsf{Chip}}$ is an unforgeable synchronized ρ-wise multi-signature that is individually correct and has correct aggregations in the presence of rogue keys and signatures, provided $\mathsf{HVC}^{\mathsf{Chip}}$ and $\mathsf{KOTS}^{\mathsf{Chip}}$ satisfy the appropriate security properties. The latter are guaranteed by Theorem 3.14 and Theorem 4.7, provided we set parameters appropriately and the appropriate Ring-SIS problems are hard. We collect the neccessary conditions in Table 2. Note that W from Theorem 5.5 corresponds to $W = \mathcal{T}_{\alpha_w}$.

6 BENCHMARKS

In this section, we define the parameters with which we instantiate Chipmunk and we provide various benchmarks, showing that our new construction significantly outperforms the previous Squirrel construction of Fleischhacker, Simkin, and Zhang [13].

Our concretely proposed construction uses the key-homomorphic one-time signature scheme for Figure 3 and $\mathsf{HVC}^{\mathsf{Chip}}$ for the homomorphic vector commitment.

6.1 Parameters and Security Estimates

The dimension of the ring $\mathcal{R}$ is fixed to $n = 512$. We choose q, q', s.t. $q, q' \equiv 1 \mod 2n$, in order speed up multiplications in $\mathcal{R}_q$ and $\mathcal{R}_{q'}$ by using NTT. The constraints that need to be satisfied by our parameters are summarized in Table 2. The concrete efficiency for a given set of parameters is determined by Table 3, which also spells out where the contributions come from. To find such concrete parameters we used a script[7], which enumerates possible parameters that satisfy all constraints, that lead to hard ring-SIS problems, and that allow for efficient NTT evaluations. For any choice of $\lambda \in \{112, 128\}$, $\rho \in \mathbb{N}$, and $\tau \in \mathbb{N}$ our script finds the parameter set that allows for the smallest possible signature size. For convenience, the results of running the script for a range of reasonable input parameters are shown in Table 6 in Appendix 7.

Let us briefly explain how our script works. First, to ensure security of Chipmunk, the specific ring-SIS problems in Theorem 3.14 need to be hard. We use the same approach to derive the security of the parameters as was used in Squirrel [13]. We adopt the so-called "realistic model" from [4]. For a BKZ of block size β, the cost in this model is estimated by $2^{0.292\beta + 16.4 + \log(\#\mathtt{SVP\ calls})}$. The LWE-estimator [3] shows that for a root Hermite factor of 1.005 we expect a block size of 286, which yields 112 bits of security under the above model. Similarly, a root Hermite factor of 1.004 yields 128 bits of security.

The output length of the hash function that is used for hashing messages needs to be large enough to prevent meet-in-the-middle type of attacks and from the security proofs we also need that for given public parameter $\boldsymbol{a}$, a fixed hash digest in $\mathcal{T}_{\alpha_H}$, and a signature $\boldsymbol{\sigma}$ for the one time signature scheme, there exists at least two short

[7] https://github.com/GottfriedHerold/Chipmunk

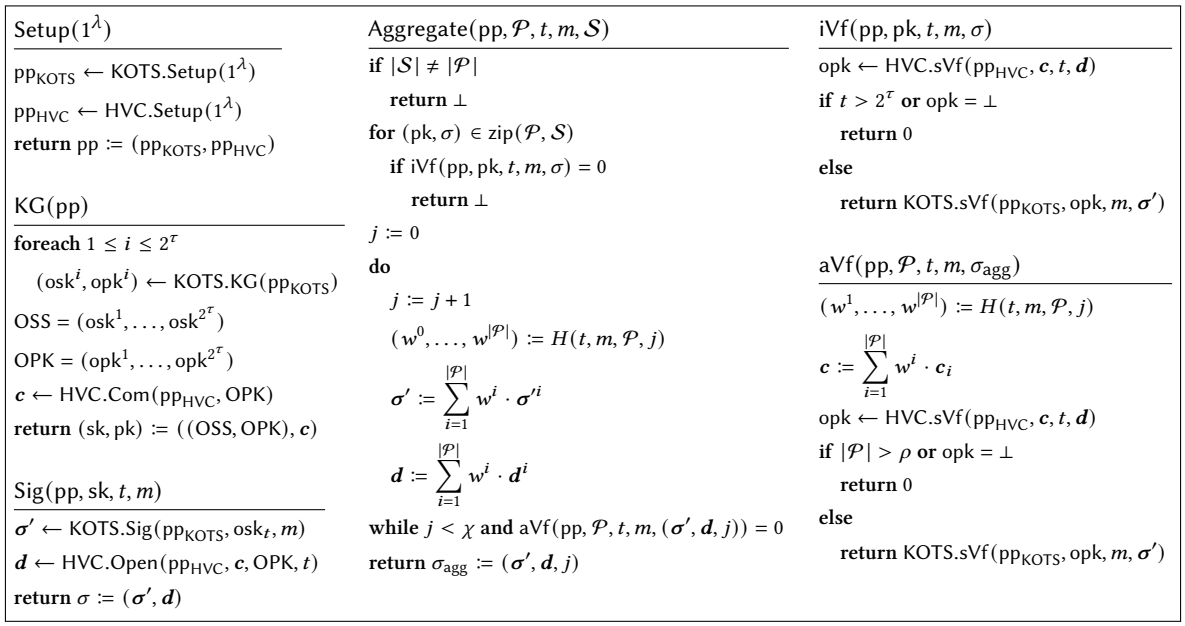
Setup(1^λ)
$\mathsf{pp}_{\mathsf{KOTS}} \leftarrow \mathsf{KOTS.Setup}(1^\lambda)$
$\mathsf{pp}_{\mathsf{HVC}} \leftarrow \mathsf{HVC.Setup}(1^\lambda)$
return $\mathsf{pp} := (\mathsf{pp}_{\mathsf{KOTS}}, \mathsf{pp}_{\mathsf{HVC}})$
KG(pp)
foreach $1 \le i \le 2^\tau$
$(\mathsf{osk}^i, \mathsf{opk}^i) \leftarrow \mathsf{KOTS.KG}(\mathsf{pp}_{\mathsf{KOTS}})$
$\mathsf{OSS} = (\mathsf{osk}^1, \ldots, \mathsf{osk}^{2^\tau})$
$\mathsf{OPK} = (\mathsf{opk}^1, \ldots, \mathsf{opk}^{2^\tau})$
$\boldsymbol{c} \leftarrow \mathsf{HVC.Com}(\mathsf{pp}_{\mathsf{HVC}}, \mathsf{OPK})$
return $(\mathsf{sk}, \mathsf{pk}) := ((\mathsf{OSS}, \mathsf{OPK}), \boldsymbol{c})$
Sig(pp, sk, t, m)
$\boldsymbol{\sigma}' \leftarrow \mathsf{KOTS.Sig}(\mathsf{pp}_{\mathsf{KOTS}}, \mathsf{osk}_t, m)$
$\boldsymbol{d} \leftarrow \mathsf{HVC.Open}(\mathsf{pp}_{\mathsf{HVC}}, \boldsymbol{c}, \mathsf{OPK}, t)$
return $\sigma := (\boldsymbol{\sigma}', \boldsymbol{d})$
Aggregate(pp, $\mathcal{P}$, t, m, $\mathcal{S}$)
if $|\mathcal{S}| \neq |\mathcal{P}|$
return $\perp$
for $(\mathsf{pk}, \sigma) \in \mathsf{zip}(\mathcal{P}, \mathcal{S})$
if $\mathsf{iVf}(\mathsf{pp}, \mathsf{pk}, t, m, \sigma) = 0$
return $\perp$
$j := 0$
do
$j := j + 1$
$(w^0, \ldots, w^{|\mathcal{P}|}) := H(t, m, \mathcal{P}, j)$
$\boldsymbol{\sigma}' := \sum_{i=1}^{|\mathcal{P}|} w^i \cdot \boldsymbol{\sigma}'^i$
$\boldsymbol{d} := \sum_{i=1}^{|\mathcal{P}|} w^i \cdot \boldsymbol{d}^i$
while $j < \chi$ and $\mathsf{aVf}(\mathsf{pp}, \mathcal{P}, t, m, (\boldsymbol{\sigma}', \boldsymbol{d}, j)) = 0$
return $\sigma_{\mathsf{agg}} := (\boldsymbol{\sigma}', \boldsymbol{d}, j)$
iVf(pp, pk, t, m, σ)
$\mathsf{opk} \leftarrow \mathsf{HVC.sVf}(\mathsf{pp}_{\mathsf{HVC}}, \boldsymbol{c}, t, \boldsymbol{d})$
if $t > 2^\tau$ or $\mathsf{opk} = \perp$
return 0
else
return $\mathsf{KOTS.sVf}(\mathsf{pp}_{\mathsf{KOTS}}, \mathsf{opk}, m, \boldsymbol{\sigma}')$
aVf(pp, $\mathcal{P}$, t, m, σ_{agg})
$(w^1, \ldots, w^{|\mathcal{P}|}) := H(t, m, \mathcal{P}, j)$
$\boldsymbol{c} := \sum_{i=1}^{|\mathcal{P}|} w^i \cdot \boldsymbol{c}_i$
$\mathsf{opk} \leftarrow \mathsf{HVC.sVf}(\mathsf{pp}_{\mathsf{HVC}}, \boldsymbol{c}, t, \boldsymbol{d})$
if $|\mathcal{P}| > \rho$ or $\mathsf{opk} = \perp$
return 0
else
return $\mathsf{KOTS.sVf}(\mathsf{pp}_{\mathsf{KOTS}}, \mathsf{opk}, m, \boldsymbol{\sigma}')$

Figure 4: A synchronized multi-signature scheme based on homomorphic vector commitments and key-homomorphic one-time signatures.

#	Source	Constraint
1	Theorem 3.14	$\beta_{\mathsf{agg}} \geq \eta\sqrt{2\alpha_w\rho(\ln\frac{2n}{\varepsilon} + \ln(2\tau\kappa + \xi\kappa'))}$
2	Theorem 3.14	$\kappa = \lceil \log_{2\eta+1} q \rceil$
3	Theorem 3.14	$\kappa' = \lceil \log_{2\eta+1} q' \rceil$
4	Theorem 4.7	$\beta_\sigma \geq 4\varphi\alpha_H\sqrt{\frac{1}{2}\alpha_w\rho \cdot \ln\frac{2n\gamma}{\varepsilon}}$
5	Theorem 4.7	$\lvert\mathcal{T}_{\alpha_H}\rvert \geq 2^{2\lambda}$
6	Theorem 4.7	$\gamma \geq ((3\lambda+\delta)/n + \log_2 q')\log_2^{-1}(\varphi + \frac{1}{2})$
7	Theorem 4.7	$\lvert\mathcal{T}_{\alpha_H}\rvert \leq 2^{2\lambda+\delta}$
8	Theorem 4.7	$q' > 16\alpha_w\alpha_H\varphi$
9	Definition 5.6	$\xi = 2$
10	Theorem 5.5	$\chi \geq \lambda/\log(\frac{1}{2\varepsilon})$
11	Theorem 5.5	$\lvert\mathcal{T}_{\alpha_w}\rvert \geq 2^\lambda$

Table 2: The constraints a set of Chipmunk parameters needs to satisfy to ensure that the proofs are applicable. The parameters additionally need to be chosen such that the associated Ring-SIS problems are hard.

corresponding $(\boldsymbol{s}_0, \boldsymbol{s}_1)$ with overwhelming probability. Lastly, we also require every randomizer to be unguessable, and therefore $|\mathcal{T}_{\alpha_w}| \geq 2^\lambda$.

6.2 Implementation

Chipmunk was implemented in Rust. The source code, as well as the scripts for parameter derivation are released to the open domain[7].

6.2.1 Comparison. For evaluating the performance of Chipmunk, there are two natural points of reference, which are the trivial solution of just storing a list of ρ Falcon signatures naively and using Squirrel [13], the previous state-of-the-art construction. The data for Chipmunk and Squirrel are collected over a same benchmark platform, an AMD 5900x with 24 threads and 32 Gigabytes of memory. The data for Falcon-512 is collected from the official website[8]. All three candidates are instantiated to yield 112 bits security.

In Table 4, we compare the three solutions for a fixed parameters set, where we only change number of signatures that are being aggregated. The comparison with the trivial Falcon solution is quite straightforward as the size of the naive solution's aggregated signatures grows linearly in the number of signers. For an aggregate signature involving 8192 signers, Chipmunk outperforms the trivial solution by a factor of 340 in terms of aggregate signature size. Obviously, the improvement only gets larger as the number of signers increases, In comparison to Squirrel, we see that for both 1024 and 8192 aggregated signatures, our scheme is better in all metrics. There are two main obstacles, namely the key generation time and the aggregate signature size, that would prevent Squirrel from being widely deployable. Our benchmarks show that Chipmunk's key generation time is better by a factor of 9 and that the size of the aggregate signatures is smaller by a factor of 4.8.

6.2.2 Scalability in terms of τ. To investigate Chipmunk's practicality, we take a closer look at the key generation time and the aggregated signature size. For this purpose, we conducted a benchmark over a typical server that is equipped with an AMD 7773x with 64 cores and 1 Terabytes of memory. In Table 5, we show how the efficiency of Chipmunk behaves, when the tree height τ grows. Notice that this platform is different from that of Section 6.2.1 and that it more accurately simulates the computational power of real-world nodes running blockchains.

[8] https://falcon-sign.info/

Component		Size in Bits
public parameters	HVC	$n(\xi\kappa' + 2\kappa)\lceil \log q \rceil$
	KOTS	$n\gamma\lceil \log q' \rceil$
Public Key	HVC Commitment	$n\lceil \log q \rceil$
Secret Key	2^τ KOTS Keys	(may regenerate on the fly)[9]
Signatures (Individual)	KOTS Signature	$n\gamma\lceil \log(4\varphi\alpha_H + 1) \rceil$
	HVC Opening	$(\tau\kappa + \xi\kappa')n\lceil \log(2\eta + 1) \rceil$
Aggregate Signatures	Agg. KOTS Signature	$n\gamma\lceil \log(2\beta_\sigma + 1) \rceil$
	Agg. HVC Opening	$(2\tau\kappa + \xi\kappa')n\lceil \log(2\beta_{\text{agg}} + 1) \rceil$
	Index j of Aggregation Attempt	$\lceil \log \chi \rceil$

Table 3: Space efficiency of Chipmunk as a function of the tunable parameters. As before $\kappa = \lceil \log_{2\eta+1} q \rceil$ and $\kappa' = \lceil \log_{2\eta+1} q' \rceil$

# signers		Falcon	Squirrel	Chipmunk	Imp. Falcon	Imp. Squirrel
	Key Generation	8.6 ms	4 min	26.5 sec	-	9.0×
	Signing	0.17 ms	2.1 ms	0.4 ms	-	5.2×
	Fresh Sig. Size	666 Bytes	45 KB	32 KB	-	1.4×
1024	Aggregation	-	1.2 sec	0.55 sec	-	2.2×
	Batch Verification	36.7 ms	19.5 ms	7.2 ms	5.1×	2.7×
	Agg. Sig. size	682 KB	572 KB	142 KB	4.8×	4.0×
8192	Aggregation	-	9.6 sec	4.3 sec	-	2.2×
	Batch Verification	294 ms	53 ms	47 ms	6.2×	1.1×
	Agg. Sig. size	5.5 MB	762 KB	160 KB	340×	4.8×

Table 4: Comparison of Chipmunk, Squirrel, and a trivial solution of just concatenating all individual signer's Falcon-512 signatures. Squirrel and Chipmunk are instantiated with $\lambda = 112$ and $\tau = 21$.

Tree Height	Key Generation	Online Signing[9]	# Signers	Aggregation	Verification	Agg. Sig. Size
21	7.4 sec	0.40 ms	1024	491 ms	7.3 ms	142 KB
			8192	3.5 sec	42.3 ms	160 KB
24	1 min	0.44 ms	1024	492 ms	7.5 ms	160 KB
			8192	3.8 sec	42.1 ms	180 KB
26	4 min	0.44 ms	1024	518 ms	7.1 ms	172 KB
			8192	4.0 sec	42.9 ms	194 KB

Table 5: Benchmark results

For the largest parameter set with $\tau = 26$ we can generate a keypair in just 4 minutes, significantly improving over the 2 hour key generation time of Squirrel. Such a key would be sufficient for 21 years of usage assuming each block takes 10 seconds to finalize as in the case of Ethereum. We conclude that the key generation time is no longer a bottleneck for practical deployment.

[9]Similar to Squirrel [13], Chipmunk is an online-offline signature scheme, since the opening of the vector commitment can be computed ahead of time without knowing the message to be signed. This means that online signing only consists of computing the one-time signature. The secret key of Chipmunk is a large tree, which can be re-derived from a pseudorandom seed whenever needed. This is computationally quite expensive, but a signer can trade storage size against offline signing speed by caching the top layers of the tree. The reported times for signing correspond to the *online* signing time.

6.2.3 One Aggregate Signature under the Microscope. To better understand the size of Chipmunk aggregate signatures, we also inspected the sizes of the aggregate signature's individual components. For the sake of concreteness, let us just consider $\tau = 21$ and $\rho = 1024$. An aggregated signature of size 142 Kilobytes, consists of the following components. The aggregated path and its adjacent nodes belonging to the homomorphic vector commitment, i.e. $2\tau\kappa$ polynomials in $\mathcal{R}$ with an infinity norm bound $\beta_{\text{agg}} < 2^{15}$, which constitute around 126 kilobytes. The aggregated decomposed public keys for the one time signature scheme, i.e., $2\kappa'$ polynomials in $\mathcal{R}$ with norm bound $\beta_{\text{agg}} < 2^{15}$ that constitute around 8 kilobytes. The aggregated one time signature, i.e., γ polynomials in $\mathcal{R}$ with norm bound $\beta_\sigma < 2^{20}$, that constitutes 8 kilobytes. Additionally we need to store the aggregation index j comprising a negligible 4 bits.

7 CONCRETE PARAMETERS

We used a script[10] to find concrete parameters that allow for instantiating Chipmunk based on a hard ring-SIS problem. We have used a fixed ring dimension $n = 512$ and a fixed error bound $\varepsilon = 2^{-15}$. For convenience a selection of possible parameter choices is given in Table 6.

ACKNOWLEDGMENTS

Nils Fleischhacker was supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2092 CASA - 390781972.

REFERENCES

[1] Jae Hyun Ahn, Matthew Green, and Susan Hohenberger. 2010. Synchronized aggregate signatures: new definitions, constructions and applications. In *ACM CCS 2010: 17th Conference on Computer and Communications Security*, Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov (Eds.). ACM Press, Chicago, Illinois, USA, 473–484. https://doi.org/10.1145/1866307.1866360

[2] Miklós Ajtai. 1999. Generating Hard Instances of the Short Basis Problem. In *ICALP 99: 26th International Colloquium on Automata, Languages and Programming (Lecture Notes in Computer Science, Vol. 1644)*, Jirí Wiedermann, Peter van Emde Boas, and Mogens Nielsen (Eds.). Springer, Heidelberg, Germany, Prague, Czech Republic, 1–9. https://doi.org/10.1007/3-540-48523-6_1

[3] Martin R. Albrecht, Rachel Player, and Sam Scott. 2015. On the concrete hardness of Learning with Errors. *J. Math. Cryptol.* 9, 3 (2015), 169–203. http://www.degruyter.com/view/j/jmc.2015.9.issue-3/jmc-2015-0016/jmc-2015-0016.xml

[4] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. 2016. Post-quantum Key Exchange - A New Hope. In *USENIX Security 2016: 25th USENIX Security Symposium*, Thorsten Holz and Stefan Savage (Eds.). USENIX Association, Austin, TX, USA, 327–343.

[5] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. 2003. Aggregate and Verifiably Encrypted Signatures from Bilinear Maps. In *Advances in Cryptology – EUROCRYPT 2003 (Lecture Notes in Computer Science, Vol. 2656)*, Eli Biham (Ed.). Springer, Heidelberg, Germany, Warsaw, Poland, 416–432. https://doi.org/10.1007/3-540-39200-9_26

[6] Dan Boneh and Sam Kim. 2020. One-Time and Interactive Aggregate Signatures from Lattices. https://crypto.stanford.edu/~skim13/agg_ots.pdf.

[7] Dan Boneh, Ben Lynn, and Hovav Shacham. 2001. Short Signatures from the Weil Pairing. In *Advances in Cryptology – ASIACRYPT 2001 (Lecture Notes in Computer Science, Vol. 2248)*, Colin Boyd (Ed.). Springer, Heidelberg, Germany, Gold Coast, Australia, 514–532. https://doi.org/10.1007/3-540-45682-1_30

[8] Cecilia Boschini, Akira Takahashi, and Mehdi Tibouchi. 2022. MuSig-L: Lattice-Based Multi-signature with Single-Round Online Phase. In *Advances in Cryptology – CRYPTO 2022, Part II (Lecture Notes in Computer Science, Vol. 13508)*, Yevgeniy Dodis and Thomas Shrimpton (Eds.). Springer, Heidelberg, Germany, Santa Barbara, CA, USA, 276–305. https://doi.org/10.1007/978-3-031-15979-4_10

[9] Ivan Damgård, Claudio Orlandi, Akira Takahashi, and Mehdi Tibouchi. 2021. Two-Round n-out-of-n and Multi-signatures and Trapdoor Commitment from Lattices. In *PKC 2021: 24th International Conference on Theory and Practice of Public Key Cryptography, Part I (Lecture Notes in Computer Science, Vol. 12710)*, Juan Garay (Ed.). Springer, Heidelberg, Germany, Virtual Event, 99–130. https://doi.org/10.1007/978-3-030-75245-3_5

[10] Manu Drijvers, Sergey Gorbunov, Gregory Neven, and Hoeteck Wee. 2020. Pixel: Multi-signatures for Consensus. In *USENIX Security 2020: 29th USENIX Security Symposium*, Srdjan Capkun and Franziska Roesner (Eds.). USENIX Association, 2093–2110.

[11] Rachid El Bansarkhani and Jan Sturm. 2016. An Efficient Lattice-Based Multisignature Scheme with Applications to Bitcoins. In *CANS 16: 15th International Conference on Cryptology and Network Security (Lecture Notes in Computer Science, Vol. 10052)*, Sara Foresti and Giuseppe Persiano (Eds.). Springer, Heidelberg, Germany, Milan, Italy, 140–155. https://doi.org/10.1007/978-3-319-48965-0_9

[12] Nils Fleischhacker, Gottfried Herold, Mark Simkin, and Zhenfei Zhang. 2023. Chipmunk: Better Synchronized Multi-Signatures from Lattices. Cryptology ePrint Archive. https://eprint.iacr.org/.

[13] Nils Fleischhacker, Mark Simkin, and Zhenfei Zhang. 2022. Squirrel: Efficient Synchronized Multi-Signatures from Lattices. In *ACM CCS 2022: 29th Conference on Computer and Communications Security*, Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi (Eds.). ACM Press, Los Angeles, CA, USA, 1109–1123. https://doi.org/10.1145/3548606.3560655

[14] Masayuki Fukumitsu and Shingo Hasegawa. 2019. A Tightly-Secure Lattice-Based Multisignature. In *6th ASIA Public-Key Cryptography Workshop*. Association for Computing Machinery, Auckland, New Zealand, 3–11. https://doi.org/10.1145/3327958.3329542

[15] Masayuki Fukumitsu and Shingo Hasegawa. 2020. A Lattice-Based Provably Secure Multisignature Scheme in Quantum Random Oracle Model. In *ProvSec 2020: 14th International Conference on Provable Security (Lecture Notes in Computer Science, Vol. 12505)*, Khoa Nguyen, Wenling Wu, Kwok-Yan Lam, and Huaxiong Wang (Eds.). Springer, Heidelberg, Germany, Singapore, 45–64. https://doi.org/10.1007/978-3-030-62576-4_3

[16] Craig Gentry and Zulfikar Ramzan. 2006. Identity-Based Aggregate Signatures. In *PKC 2006: 9th International Conference on Theory and Practice of Public Key Cryptography (Lecture Notes in Computer Science, Vol. 3958)*, Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin (Eds.). Springer, Heidelberg, Germany, New York, NY, USA, 257–273. https://doi.org/10.1007/11745853_17

[17] Susan Hohenberger and Brent Waters. 2018. Synchronized Aggregate Signatures from the RSA Assumption. In *Advances in Cryptology – EUROCRYPT 2018, Part II (Lecture Notes in Computer Science, Vol. 10821)*, Jesper Buus Nielsen and Vincent Rijmen (Eds.). Springer, Heidelberg, Germany, Tel Aviv, Israel, 197–229. https://doi.org/10.1007/978-3-319-78375-8_7

[18] Kazuharu Itakura and Katsuhiro Nakamura. 1983. A public-key cryptosystem suitable for digital multisignatures. *NEC Research & Development* 71 (1983), 1–8.

[19] Meenakshi Kansal and Ratna Dutta. 2020. Round Optimal Secure Multisignature Schemes from Lattice with Public Key Aggregation and Signature Compression. In *AFRICACRYPT 20: 12th International Conference on Cryptology in Africa (Lecture Notes in Computer Science, Vol. 12174)*, Abderrahmane Nitaj and Amr M. Youssef (Eds.). Springer, Heidelberg, Germany, Cairo, Egypt, 281–300. https://doi.org/10.1007/978-3-030-51938-4_14

[20] Zi-Yuan Liu, Yi-Fan Tseng, and Raylin Tso. 2020. Cryptanalysis of a round optimal lattice-based multisignature scheme. Cryptology ePrint Archive, Report 2020/1172. https://eprint.iacr.org/2020/1172.

[21] Vadim Lyubashevsky and Daniele Micciancio. 2008. Asymptotically Efficient Lattice-Based Digital Signatures. In *TCC 2008: 5th Theory of Cryptography Conference (Lecture Notes in Computer Science, Vol. 4948)*, Ran Canetti (Ed.). Springer, Heidelberg, Germany, San Francisco, CA, USA, 37–54. https://doi.org/10.1007/978-3-540-78524-8_3

[22] Changshe Ma and Mei Jiang. 2019. Practical Lattice-Based Multisignature Schemes for Blockchains. *IEEE Access* 7 (2019), 179765–179778. https://doi.org/10.1109/ACCESS.2019.2958816

[23] Colin McDiarmid. 1989. On the Method of Bounded Differences. In *Surveys in Combinatorics, 1989: Invited Papers at the Twelfth British Combinatorial Conference (London Mathematical Society Lecture Note Series, Vol. 141)*, Johannes Siemons (Ed.). Cambridge University Press, Norwich, UK, 148–188. https://doi.org/10.1017/CBO9781107359949.008

[24] Silvio Micali, Kazuo Ohta, and Leonid Reyzin. 2001. Accountable-Subgroup Multisignatures: Extended Abstract. In *ACM CCS 2001: 8th Conference on Computer and Communications Security*, Michael K. Reiter and Pierangela Samarati (Eds.). ACM Press, Philadelphia, PA, USA, 245–254. https://doi.org/10.1145/501983.502017

[25] Daniele Micciancio. 2007. Generalized Compact Knapsacks, Cyclic Lattices, and Efficient One-Way Functions. *computatinal complexity* 16, 4 (Dec. 2007), 365–411. https://doi.org/10.1007/s00037-007-0234-9

[26] Chunyan Peng and Xiujuan Du. 2020. New Lattice-Based Digital Multi-signature Scheme. In *6th International Conference of Pioneering Computer Scientists, Engineers and Educators (CCIS, Vol. 1258)*. Springer, Heidelberg, Germany, Taiyuan, China, 129–137. https://doi.org/10.1007/978-981-15-7984-4_10

[27] Peter W Shor. 1994. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th annual symposium on foundations of computer science*. Ieee, 124–134.

[10] https://github.com/GottfriedHerold/Chipmunk

Parameter Sets					KOTS Parameters					HVC Parameters			**Agg. Sig. Size**
λ	τ	ρ	α_w	χ	α_H	φ	γ	β_σ	q'	η	β_{agg}	q	(Kilobytes)
		1024	16	11	37	13	6	775,143	3,168,257	29	25,133	202,753	142
	21	8192	16	11	37	16	6	2,698,381	10,872,833	49	120,108	962,561	160
		131072	16	11	37	13	7	8,803,786	35,301,377	99	970,665	7,790,593	187
		1024	16	11	37	13	6	775,143	3,168,257	29	25,202	202,753	160
112	24	8192	16	11	37	16	6	2,698,381	10,872,833	49	120,438	964,609	180
		131072	16	11	37	13	7	8,803,786	35,301,377	99	973,331	7,790,593	211
		1024	16	11	37	13	6	775,143	3,168,257	29	25,243	202,753	172
	26	8192	16	11	37	16	6	2,698,381	10,872,833	50	123,098	995,329	194
		131072	16	11	37	13	7	8,803,786	35,301,377	99	974,934	7,806,977	227
		1024	19	13	44	9	7	698,123	2,856,961	31	29,276	249,857	144
	21	8192	19	13	44	8	8	1,761,048	7,114,753	12	32,252	270,337	189
		131072	19	13	44	9	8	7,924,716	31,776,769	17	182,757	1,492,993	224
		1024	19	13	44	9	7	698,123	2,856,961	31	29,357	249,857	162
128	24	8192	19	13	44	8	8	1,761,048	7,114,753	12	32,339	270,337	213
		131072	19	13	44	9	8	7,924,716	31,776,769	17	183,255	1,492,993	252
		1024	19	13	44	9	7	698,123	2,856,961	31	29,405	249,857	174
	26	8192	19	13	44	8	8	1,761,048	7,114,753	12	32,392	270,337	229
		131072	19	13	44	9	8	7,924,716	31,776,769	17	183,554	1,492,993	271

Table 6: Parameter sets for Chipmunk for a fixed ring dimension $n = 512$ and fixed error bound $\varepsilon = 2^{-15}$.